Improved Approximation for Node-Disjoint Paths in Grids with Sources on the Boundary

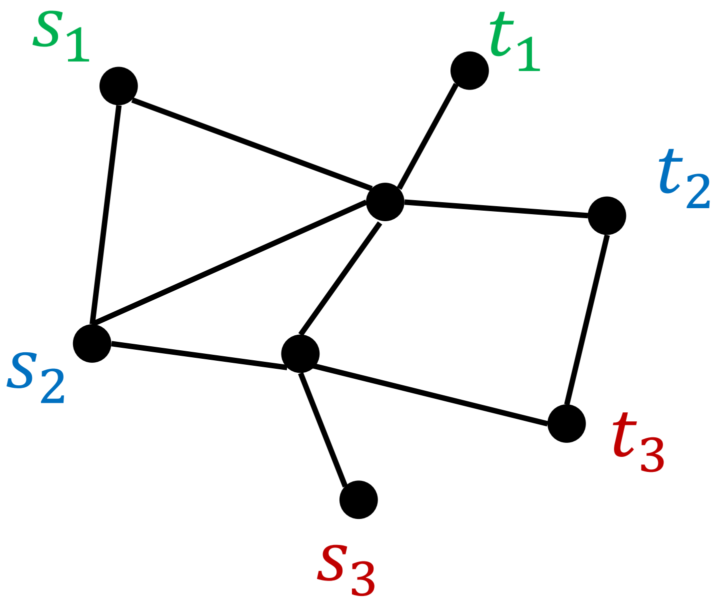
JULIA CHUZHOY DAVID KIM RACHIT NIMAVAT

TOYOTA TECHNOLOGICAL INSTITUTE AT CHICAGO

ICALP, 2018

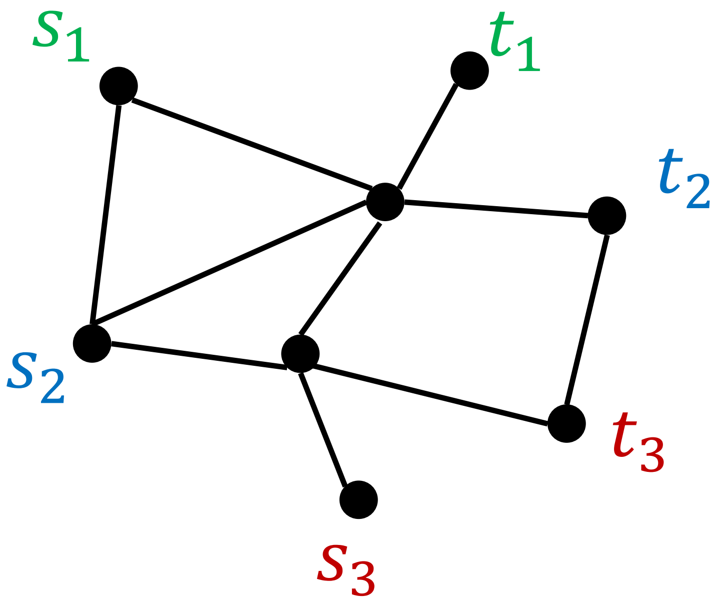
Node-Disjoint Paths (NDP)

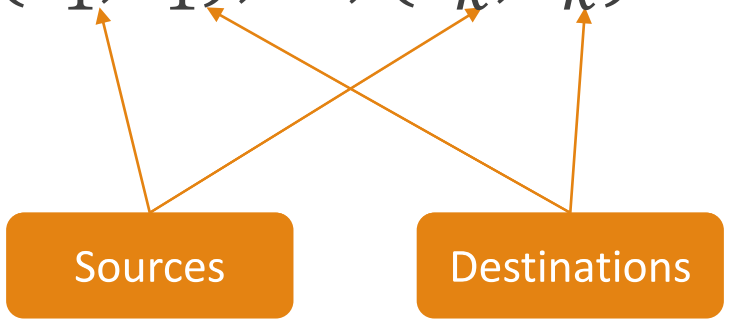
Input: Undirected graph and demand pairs $(s_1, t_1), \dots, (s_k, t_k)$



Node-Disjoint Paths (NDP)

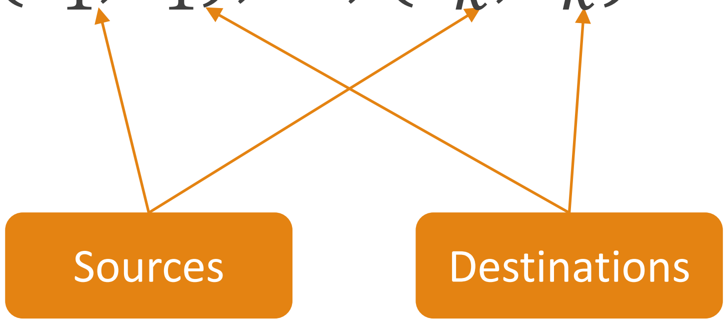
Input: Undirected graph and demand pairs $(s_1, t_1), \dots, (s_k, t_k)$

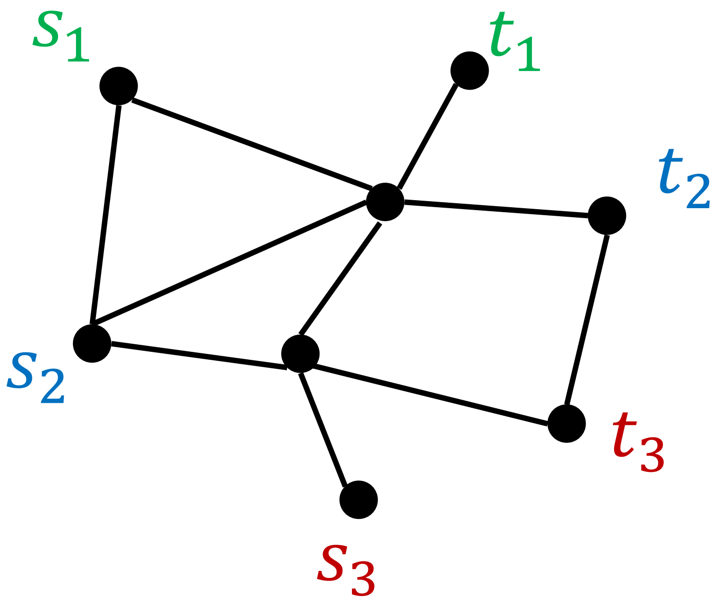




Node-Disjoint Paths (NDP)

Input: Undirected graph and demand pairs $(s_1, t_1), \ldots, (s_k, t_k)$

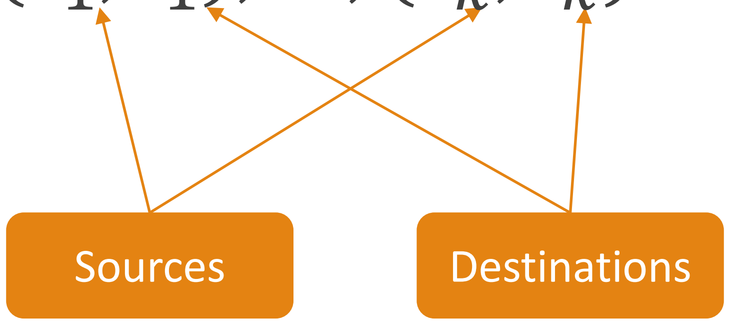


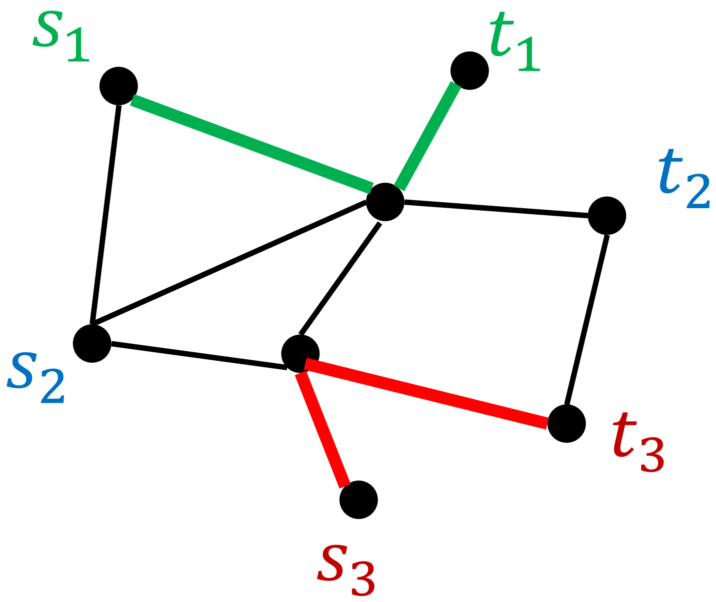


Goal: Route as many pairs as possible via node-disjoint paths

Node-Disjoint Paths (NDP)

Input: Undirected graph and demand pairs $(s_1, t_1), \dots, (s_k, t_k)$





OPT: 2

Goal: Route as many pairs as possible via node-disjoint paths

Known Results

- Constant k $\Rightarrow$ Efficient algorithm [Robertson, Seymour '90]

Known Results

- Constant $k \Rightarrow$ Efficient algorithm [Robertson, Seymour '90]
  - FPT algorithm: $f(k) \cdot n^2$ [Robertson, Seymour '90 → Kawarbayashi, Kobayashi, Reed '12]

$$f(k) = 2^{2^{2^{2^{\cdot^{\cdot^{\cdot^{(k)}}}}}}}$$

Known Results

- Constant k $\Rightarrow$ Efficient algorithm [Robertson, Seymour '90]
- k part of input $\Rightarrow$ NP-Hard [Karp '72]

Known Results

- Constant $k \Rightarrow$ Efficient algorithm [Robertson, Seymour '90]
- k part of input $\Rightarrow$ NP-Hard [Karp '72]

- $O(\sqrt{n})$ −approx. [Kolliopoulos, Stein '98]
- Roughly $\Omega\left(\sqrt{\log n}\right)$ −hardness of approx. [Andrews, Zhang '05], [Andrews, Chuzhoy, Guruswami, Khanna, Talwar, Zhang '10]

Known Results

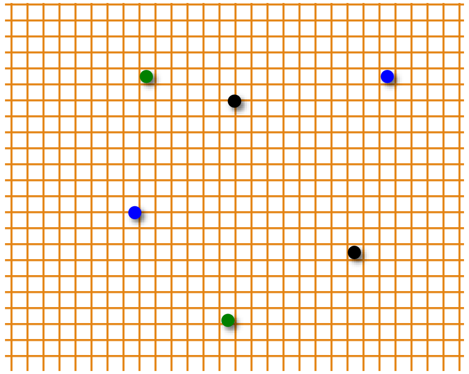
- Constant $k \Rightarrow$ Efficient algorithm [Robertson, Seymour '90]
- k part of input $\Rightarrow$ NP-Hard [Karp '72]

- $O(\sqrt{n})$ −approx. [Kolliopoulos, Stein '98]
- Roughly $\Omega\left(\sqrt{\log n}\right)$ −hardness of approx. [Andrews, Zhang '05], [Andrews, Chuzhoy, Guruswami, Khanna, Talwar, Zhang '10]

- What about *simpler* cases?
  - Analysis of $O(\sqrt{n})$-approx. algorithm is tight on grids! ☹

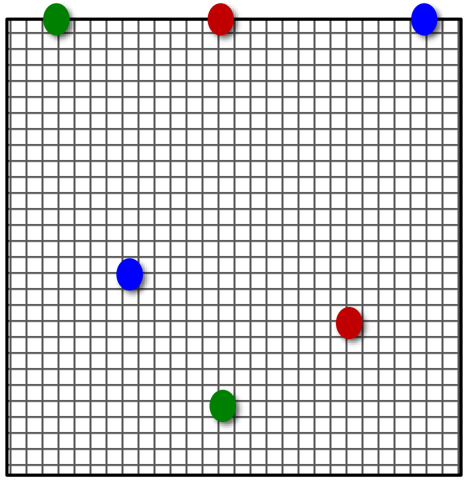
NDP-Grid

- $O(n^{1/4})$ −approx. for NDP-Grid [Chuzhoy, Kim '15]
- $n^{\Omega(1/(\log\log n)^2)}$ hardness [Chuzhoy, Kim, N. '18]



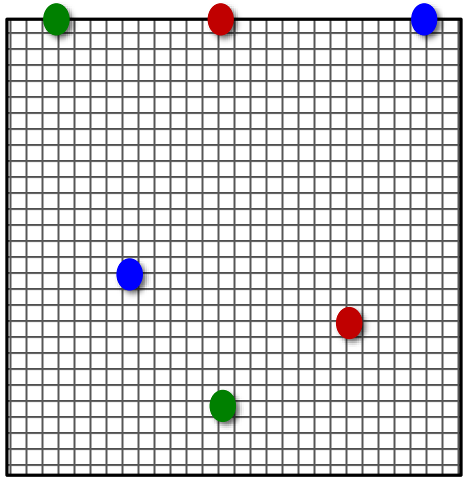
Our Result

$2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP-Grid if sources appear on the boundary

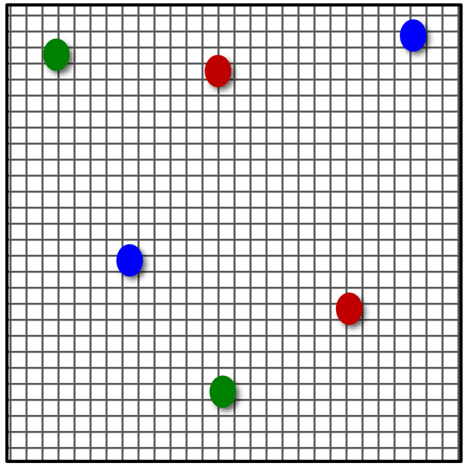


Our Result

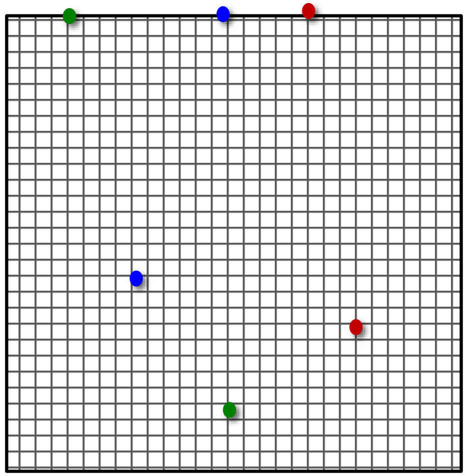
$2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP-Grid if sources appear on the boundary



$\delta \cdot 2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP-Grid if sources are at distance $\leq \delta$ from the boundary



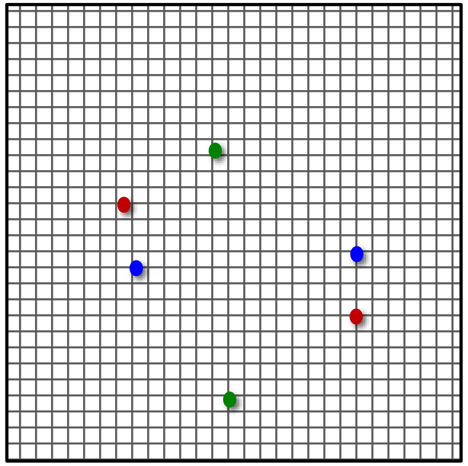
"Complementary" Results



$2^{O(\sqrt{\log n})}$ −approximation

Grid with sources on boundary

This Result

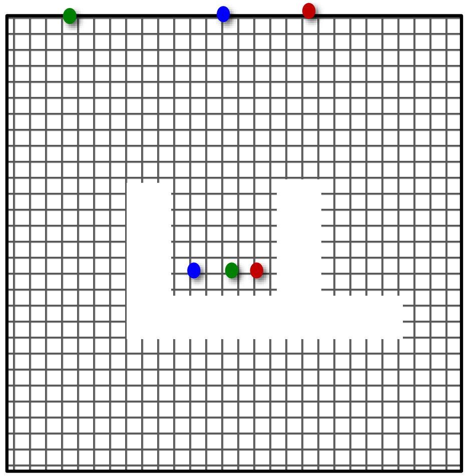


$2^{\Omega(\log^{0.99} n)}$ −hard

Grid with sources far from boundary

[Chuzhoy, Kim, N. '18]

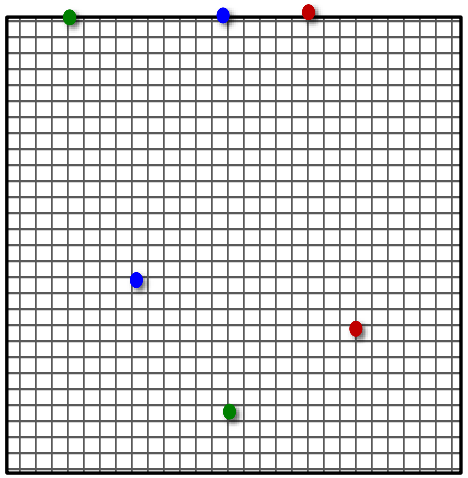
“Complementary” Results



$2^{\Omega(\sqrt{\log n})}$ −hard

Grid with holes

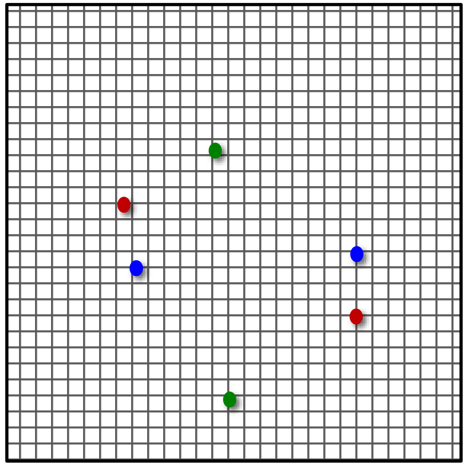
[Chuzhoy, Kim, N. ‘17]



$2^{O(\sqrt{\log n})}$ −approximation

Grid with sources on boundary

This Result



$2^{\Omega(\log^{0.99} n)}$ −hard

Grid with sources far from boundary

[Chuzhoy, Kim, N. ‘18]

Multicommodity Flow Relaxation

- A natural way to solve NDP
- Relax “integrality of flow paths” requirement

Multicommodity Flow Relaxation

- A natural way to solve NDP
- Relax "integrality of flow paths" requirement

Max. $\sum_i flow(s_i \to t_i)$

$\forall i, \quad flow(s_i \to t_i) \leq 1$
$\forall v, \quad flow_v \leq 1$

Multicommodity Flow Relaxation

- A natural way to solve NDP
- Relax "integrality of flow paths" requirement

Max. $\sum_i flow(s_i \rightarrow t_i)$

$\forall i, \quad flow(s_i \rightarrow t_i) \leq 1$
$\forall v, \quad flow_v \leq 1$

$OPT_{LP} \geq OPT$

Multicommodity Flow Relaxation

- A natural way to solve NDP
- Relax "integrality of flow paths" requirement

Max. $\sum_i flow(s_i \to t_i)$

$\forall i, \quad flow(s_i \to t_i) \leq 1$
$\forall v, \quad flow_v \leq 1$

$OPT_{LP} \geq OPT$

[Kolliopoulos, Stein '98] Approx. Algorithm

While there is a path with $flow(P) > 0$:

- Add such shortest path P to the solution
- Delete vertices of P from the graph

Multicommodity Flow Relaxation

- A natural way to solve NDP
- Relax "integrality of flow paths" requirement

Max. $\sum_i flow(s_i \to t_i)$

$\forall i, \quad flow(s_i \to t_i) \leq 1$
$\forall v, \quad flow_v \leq 1$

$OPT_{LP} \geq OPT$

[Kolliopoulos, Stein '98] Approx. Algorithm
While there is a path with $flow(P) > 0$:
- Add such shortest path P to the solution
- Delete vertices of P from the graph

$\sqrt{n}$-approximation algorithm!

$OPT_{LP} \leq \sqrt{n} \cdot OPT$

Multicommodity Flow Relaxation

- A natural way to solve NDP
- Relax "integrality of flow paths" requirement

Max. $\sum_i flow(s_i \to t_i)$

$\forall i, \quad flow(s_i \to t_i) \leq 1$
$\forall v, \quad flow_v \leq 1$

$OPT_{LP} \geq OPT$

[Kolliopoulos, Stein '98] Approx. Algorithm
While there is a path with $flow(P) > 0$:
- Add such shortest path P to the solution
- Delete vertices of P from the graph

$\sqrt{n}$-approximation algorithm!

$OPT_{LP} \leq \sqrt{n} \cdot OPT$

On grid with sources and destinations on boundary, integrality gap is $\Omega(\sqrt{n})$ ☹

Multicommodity Flows: Is That It?

- On grid with sources and destinations close to boundary, integrality gap is $\Omega(\sqrt{n})$ ☹
  - But DP works in this regime ☺
  - $O(n^{1/4})$-approx for NDP-Grid [Chuzhoy, Kim '15]

Multicommodity Flows: Is That It?

- On grid with sources and destinations close to boundary, integrality gap is $\Omega(\sqrt{n})$ ☹
  - But DP works in this regime ☺
  - $O(n^{1/4})$-approx for NDP-Grid [Chuzhoy, Kim '15]

- Even when sources and destinations are far boundary, integrality gap remains $\Omega(n^{1/8})$ ☹ [Chuzhoy, Kim '15]

Beyond Multicommodity Flows

1. Write a LP to ***select*** a *good* set of demand pairs
2. Use a separate combinatorial algorithm for routing

Beyond Multicommodity Flows

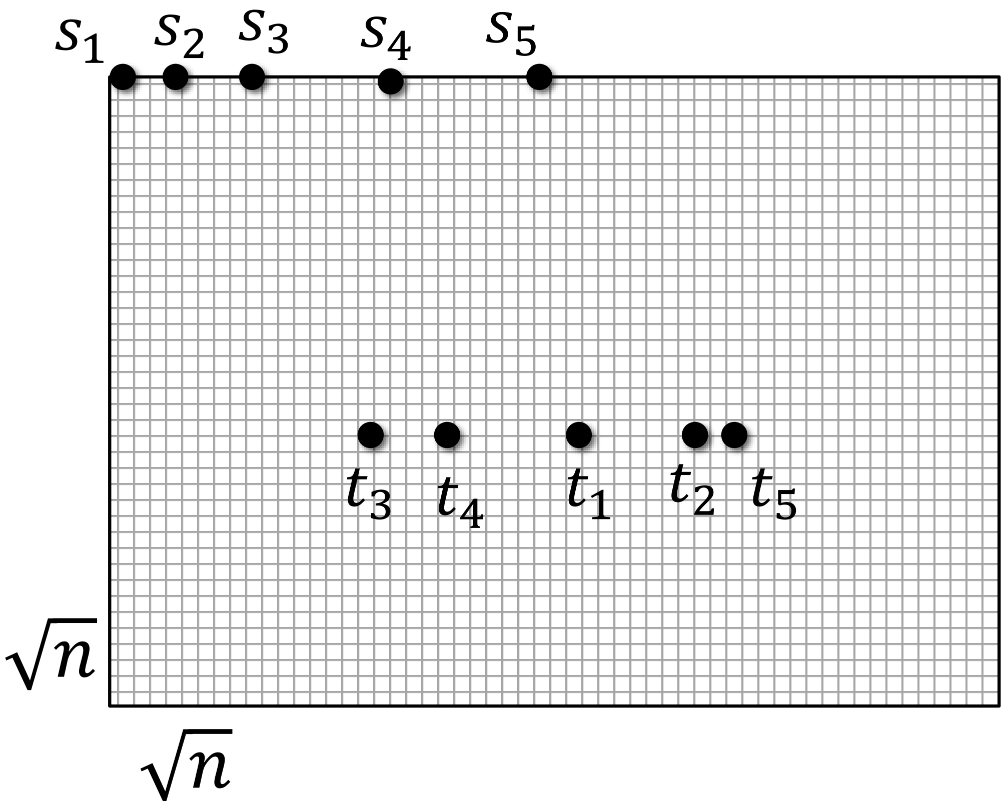
1. Write a LP to ***select*** a *good* set of demand pairs
2. Use a separate combinatorial algorithm for routing

Beyond Multicommodity Flows

1. Write a LP to ***select*** a *good* set of demand pairs
2. Use a separate combinatorial algorithm for routing

Assume for simplicity:

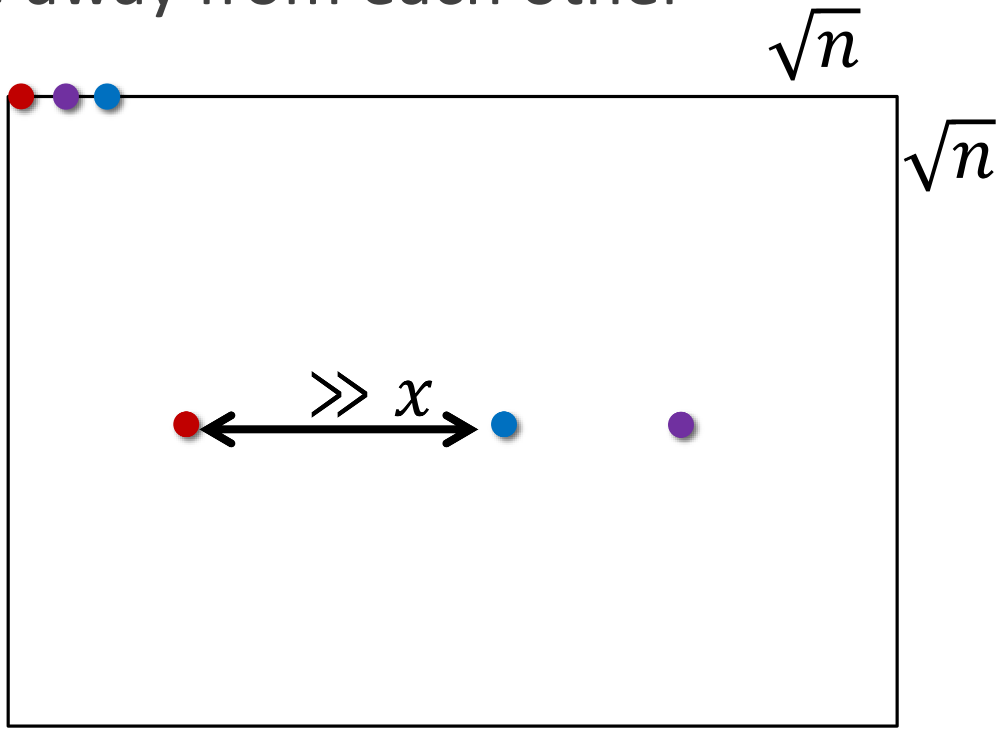
- All sources and destinations are distinct
- All sources lie on top boundary
- All destinations lie on a ***single row*** at distance $\gg OPT$ from grid boundaries



Routing

Assume we have:

- x demand pairs
- Their destinations are at $\gg x$ distance away from each other



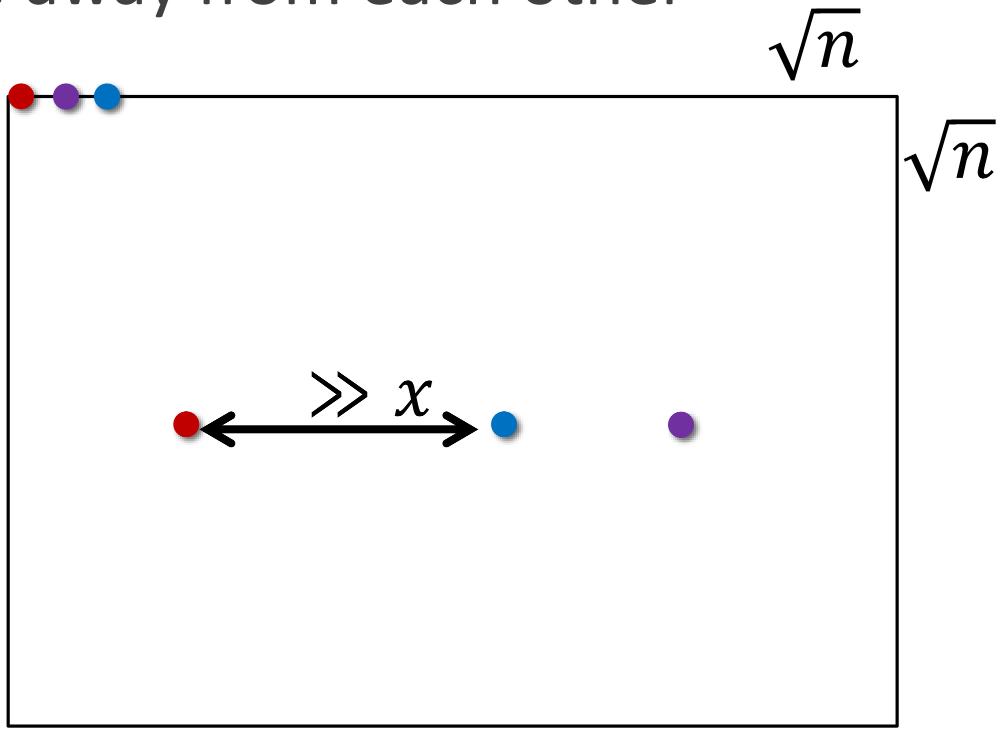
Routing

Assume we have:

- x demand pairs
- Their destinations are at $\gg x$ distance away from each other

Snake-like Routing

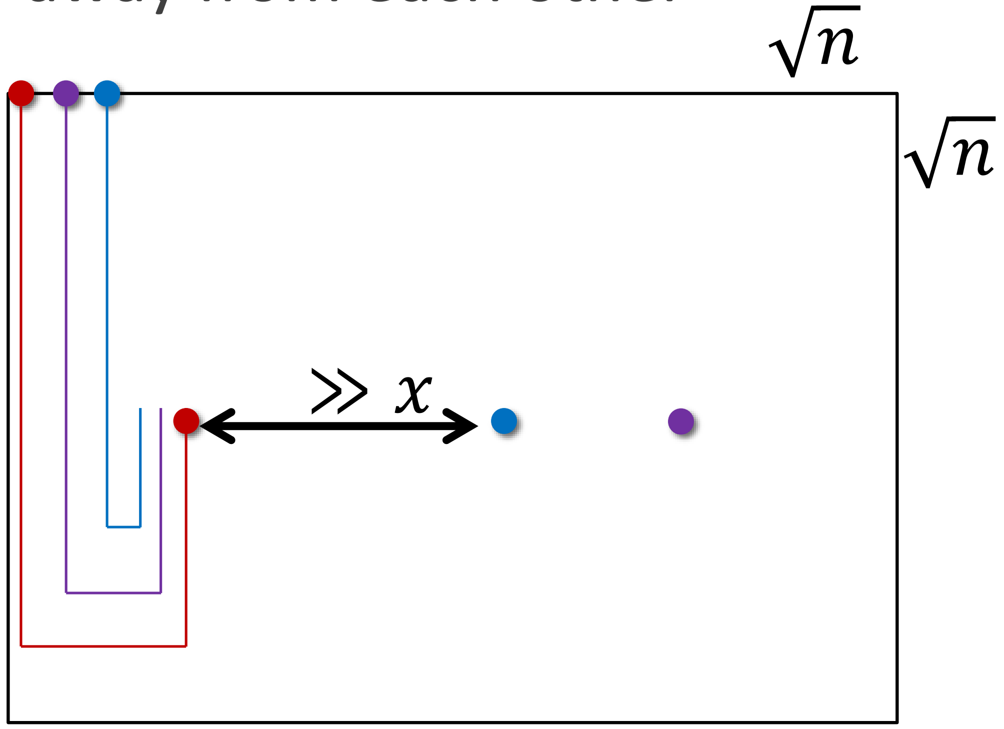
Used in [Chuzhoy, Kim '15] and [Cutler, Shiloach '78]



Routing

Assume we have:

- x demand pairs
- Their destinations are at $\gg x$ distance away from each other



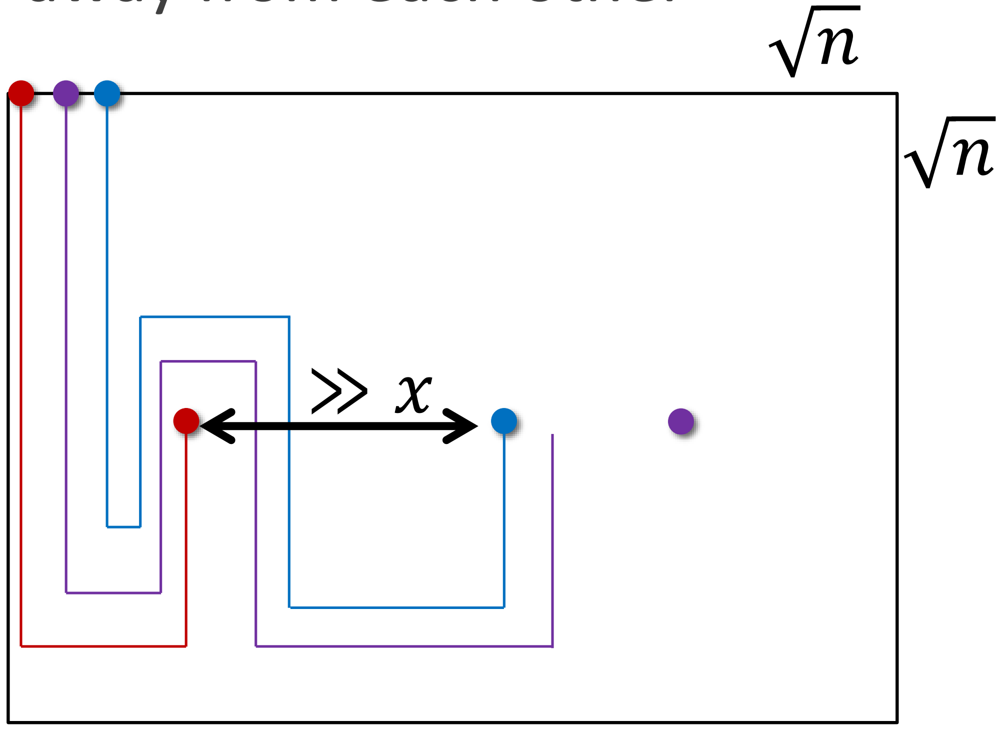
Snake-like Routing

Used in [Chuzhoy, Kim '15] and [Cutler, Shiloach '78]

Routing

Assume we have:

- x demand pairs
- Their destinations are at $\gg x$ distance away from each other



Snake-like Routing

Used in [Chuzhoy, Kim '15] and [Cutler, Shiloach '78]

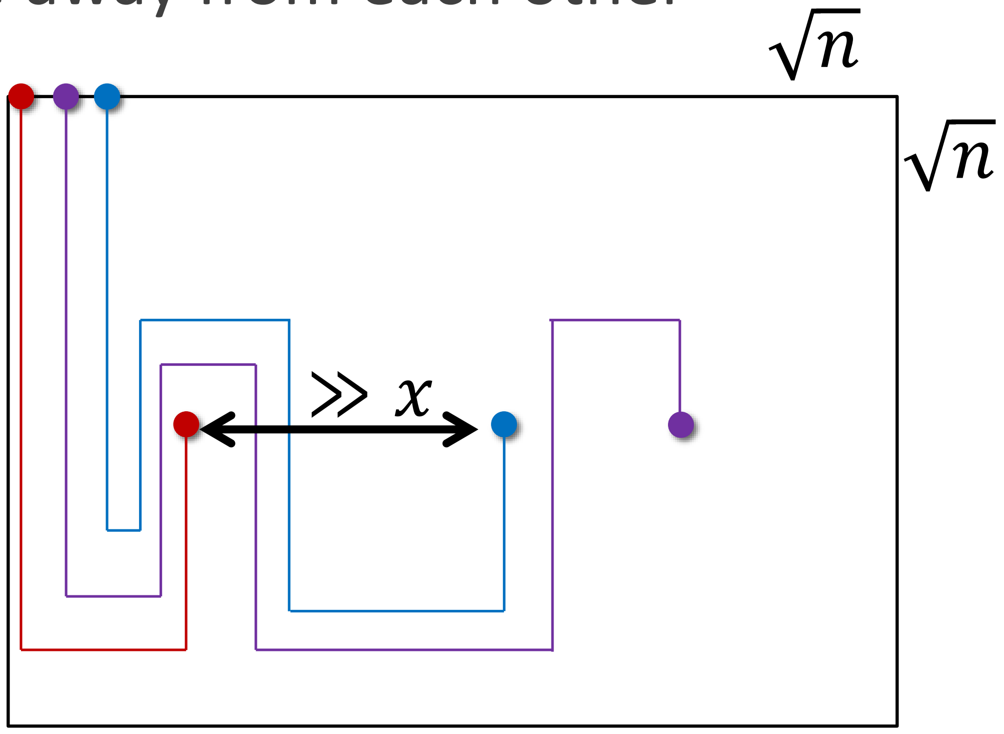
Routing

Assume we have:

- x demand pairs
- Their destinations are at $\gg x$ distance away from each other

Snake-like Routing

Used in [Chuzhoy, Kim '15]
and [Cutler, Shiloach '78]



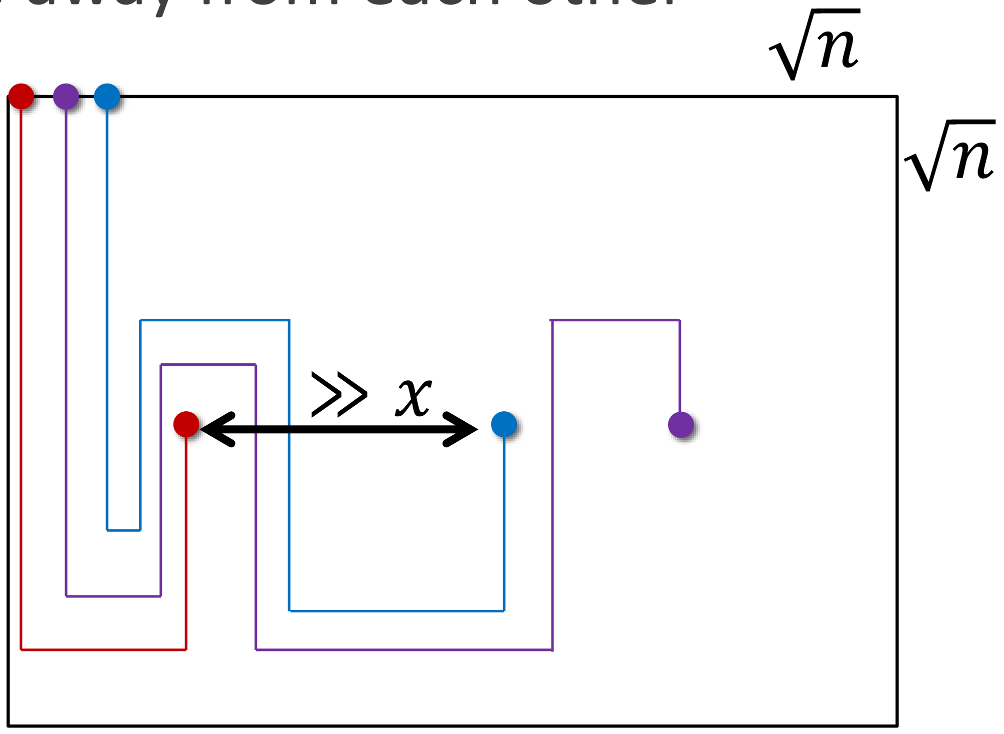
Routing

Assume we have:

- x demand pairs
- Their destinations are at $\gg x$ distance away from each other

- To route $n^{1/4}$ demand pairs, need $\approx n^{1/4}$ spacing
- Can't route more than $n^{1/4}$ demand pairs
- But OPT can be $\approx \sqrt{n}$

Looks very inefficient…



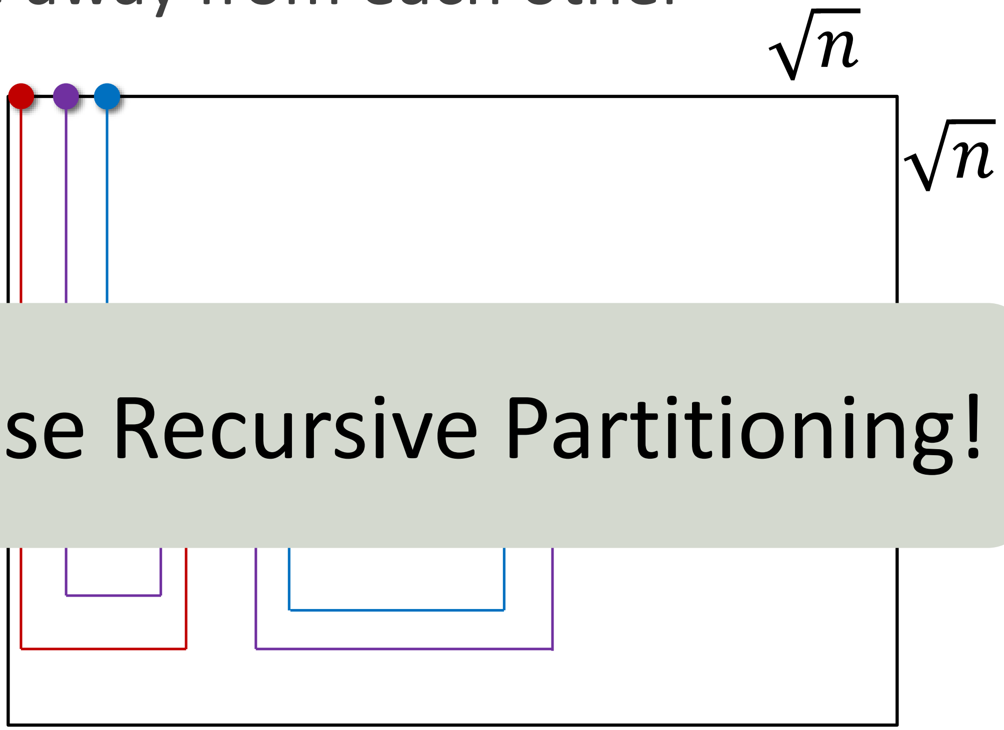
Routing

Assume we have:

- x demand pairs
- Their destinations are at $\gg x$ distance away from each other

- To route $n^{1/4}$ demand pairs, need $\approx n^{1/4}$ spacing
- Can't route more than $n^{1/4}$ demand pairs
- But OPT can be $\approx \sqrt{n}$

Looks very inefficient...



Use Recursive Partitioning!

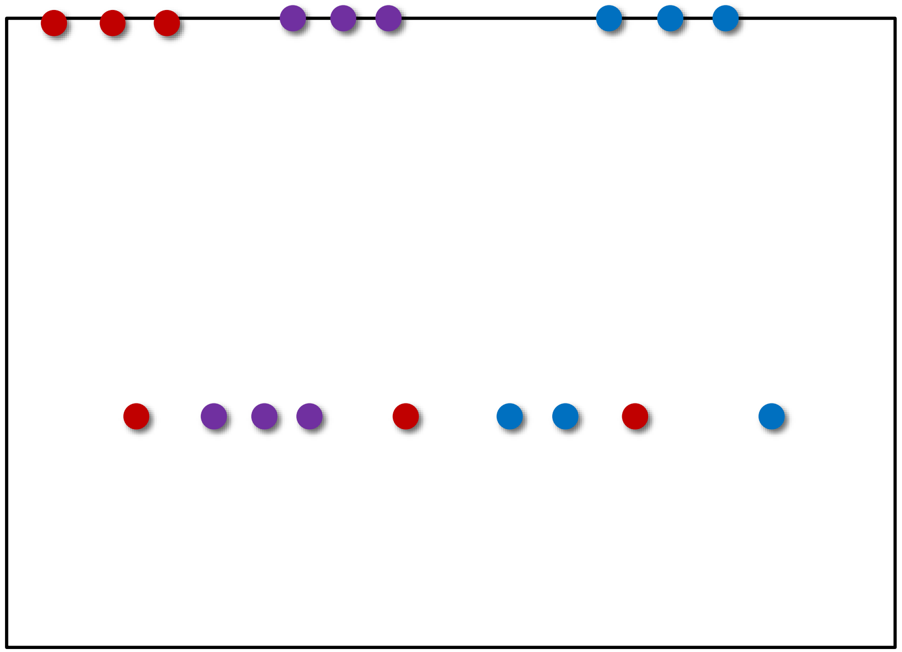
Recursive Partitioning

Don’t really need ***all*** the destinations to be that far from each other

Recursive Partitioning

Don’t really need ***all*** the destinations to be that far from each other

Def^n: A set of x demand pairs and its coloring by η colors:

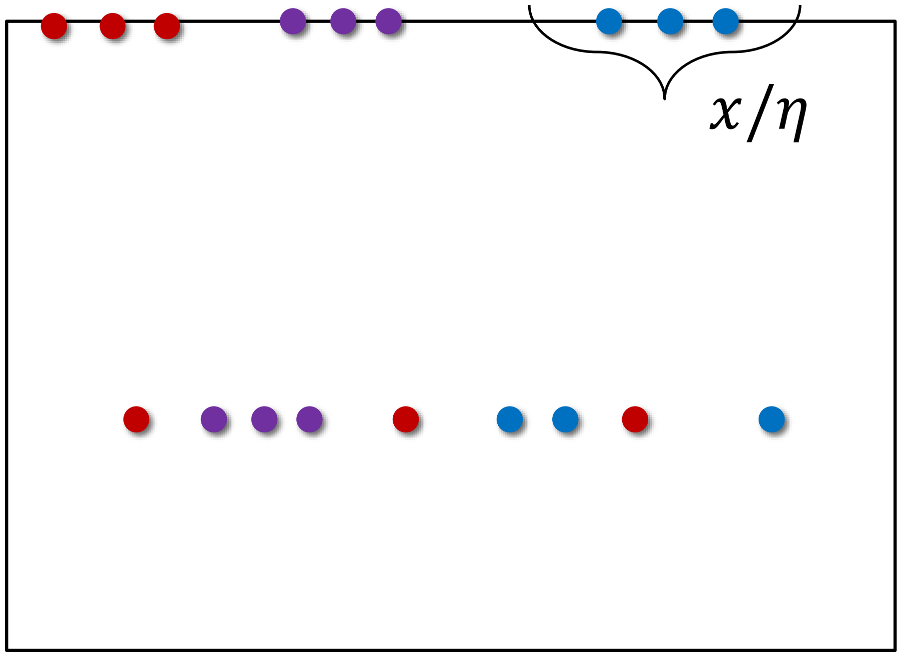


Recursive Partitioning

Don't really need ***all*** the destinations to be that far from each other

Def^{n}: A set of x demand pairs and its coloring by η colors:

- x/η demand pairs of each color

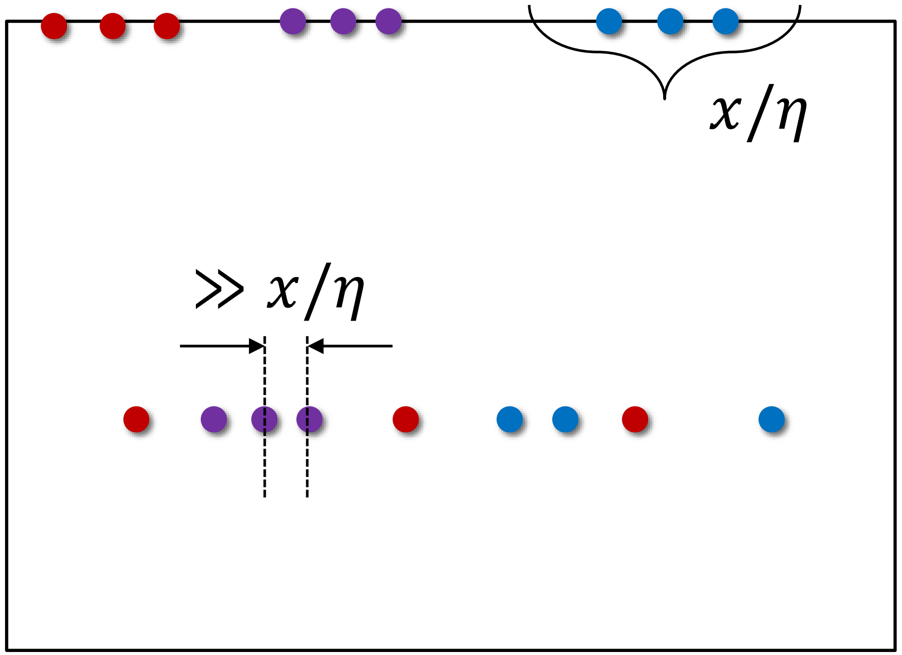


Recursive Partitioning

Don't really need ***all*** the destinations to be that far from each other

Def^{n}: A set of x demand pairs and its coloring by η colors:

- x/η demand pairs of each color
- All destinations are at distance $\gg x/\eta$ from each other

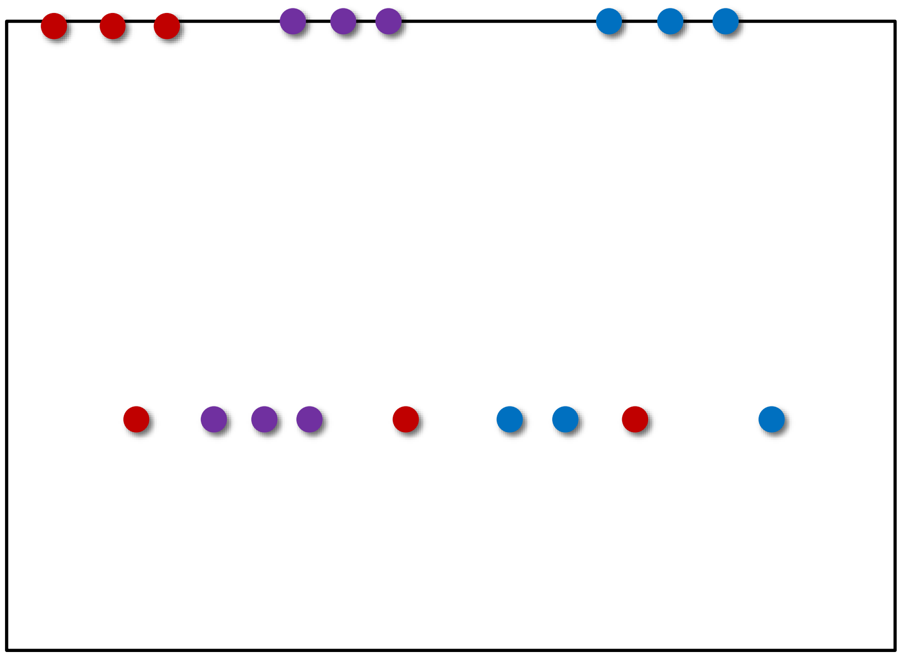


Recursive Partitioning

Don’t really need ***all*** the destinations to be that far from each other

Def^n: A set of x demand pairs and its coloring by η colors:

- x/η demand pairs of each color
- All destinations are at distance $\gg x/\eta$ from each other
- Single *source-interval* for each color

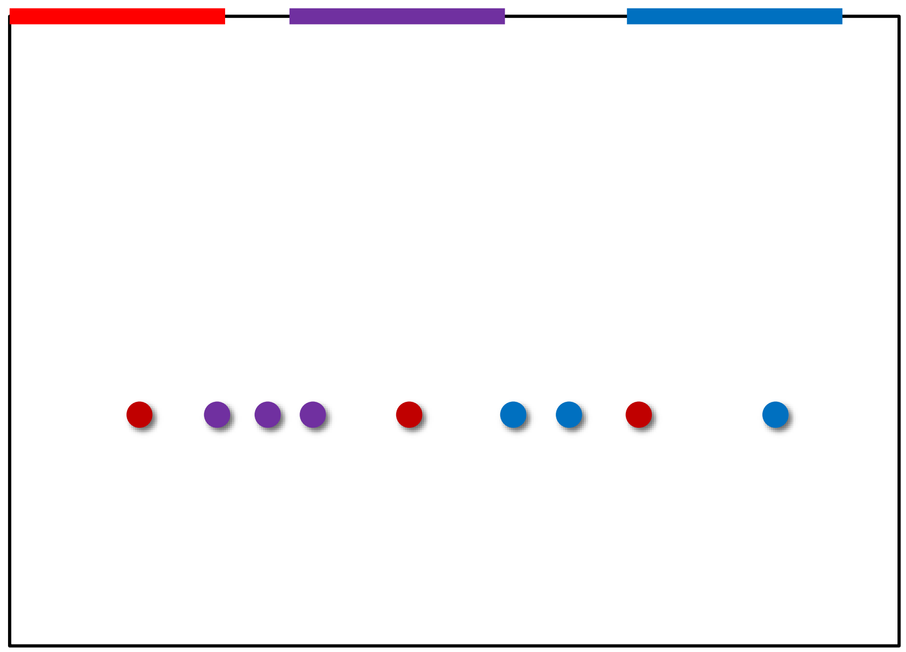


Recursive Partitioning

Don't really need ***all*** the destinations to be that far from each other

Def^{n}: A set of x demand pairs and its coloring by η colors:

- x/η demand pairs of each color
- All destinations are at distance $\gg x/\eta$ from each other
- Single *source-interval* for each color

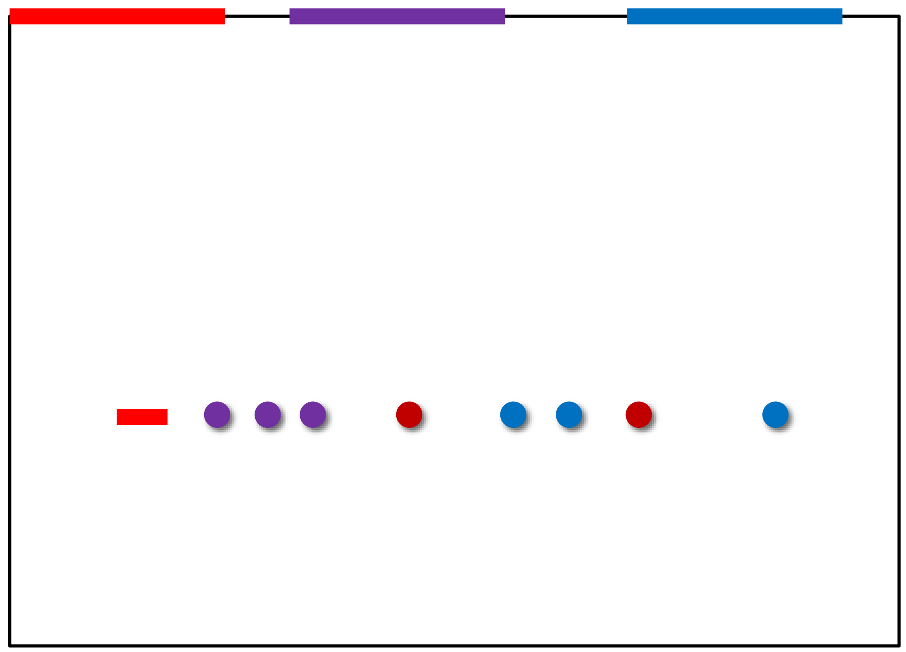


Recursive Partitioning

Don't really need ***all*** the destinations to be that far from each other

Def^{n}: A set of x demand pairs and its coloring by η colors:

- x/η demand pairs of each color
- All destinations are at distance $\gg x/\eta$ from each other
- Single *source-interval* for each color

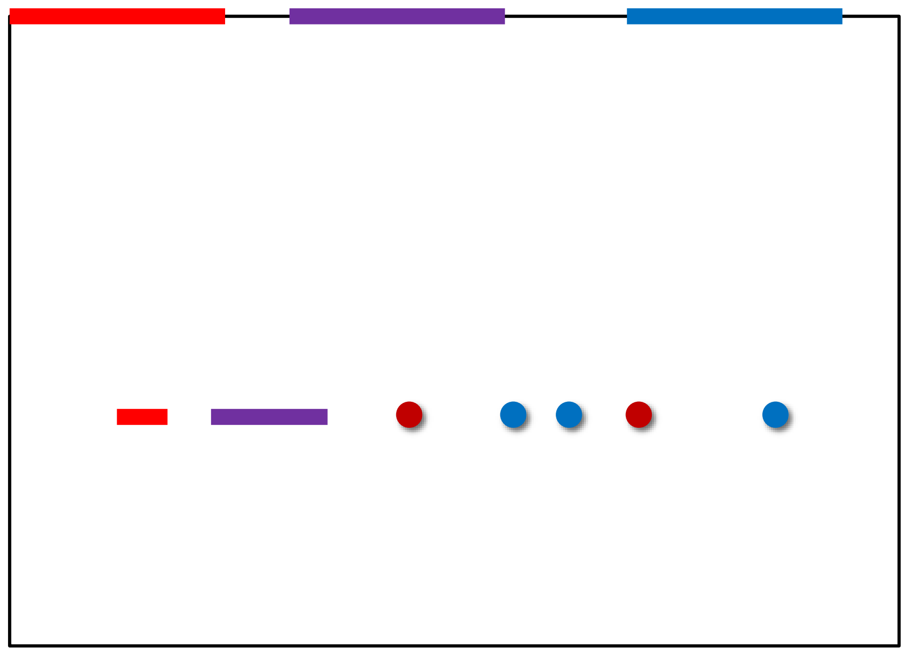


Recursive Partitioning

Don't really need ***all*** the destinations to be that far from each other

Def^{n}: A set of x demand pairs and its coloring by η colors:

- x/η demand pairs of each color
- All destinations are at distance $\gg x/\eta$ from each other
- Single *source-interval* for each color

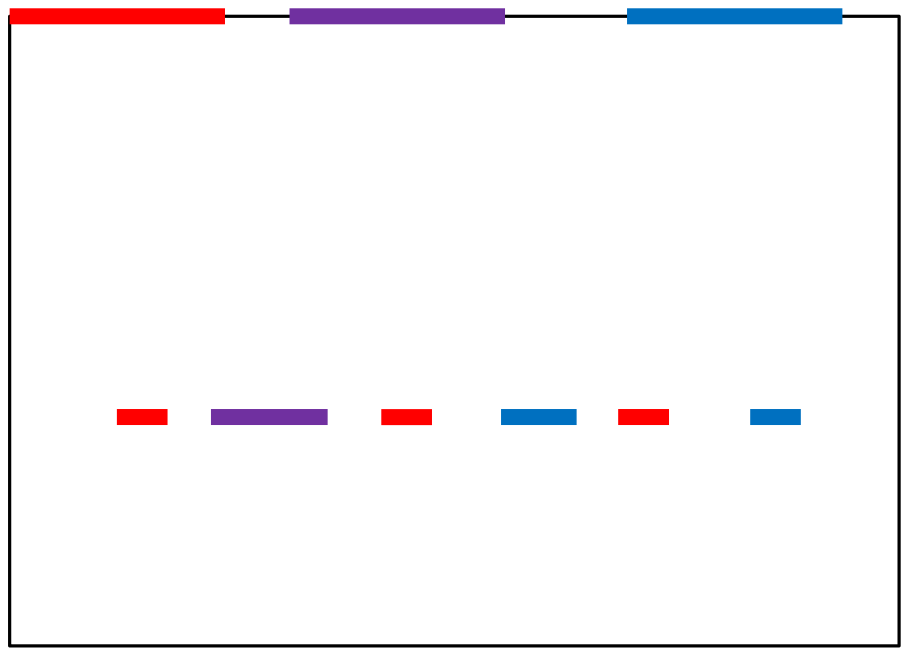


Recursive Partitioning

Don't really need ***all*** the destinations to be that far from each other

Def^{n}: A set of x demand pairs and its coloring by η colors:

- x/η demand pairs of each color
- All destinations are at distance $\gg x/\eta$ from each other
- Single *source-interval* for each color

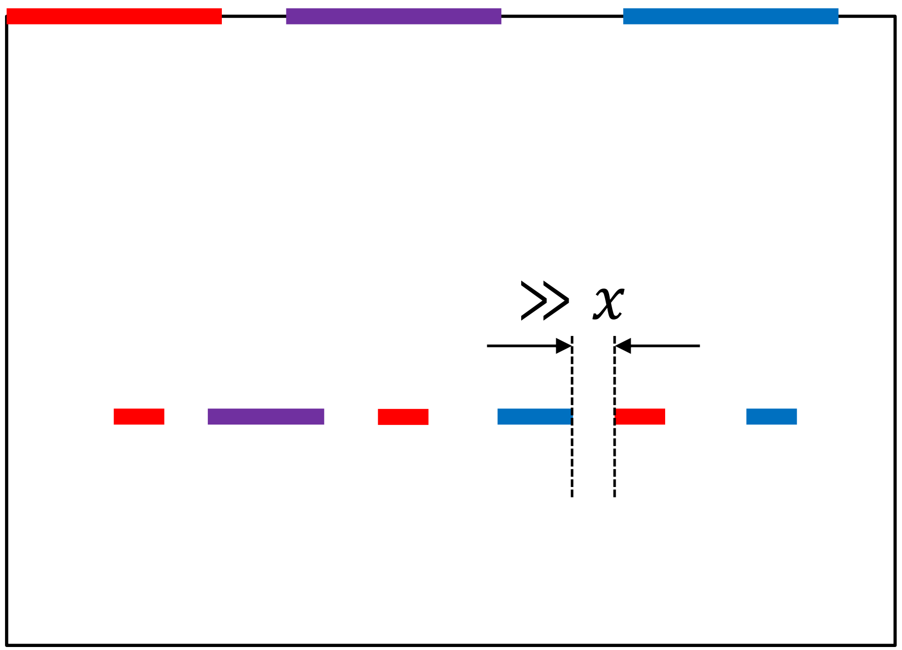


Recursive Partitioning

Don't really need ***all*** the destinations to be that far from each other

Def^n: A set of x demand pairs and its coloring by η colors:

- x/η demand pairs of each color
- All destinations are at distance $\gg x/\eta$ from each other
- Single *source-interval* for each color
- Destination-intervals are at distance $\gg x$ from each other

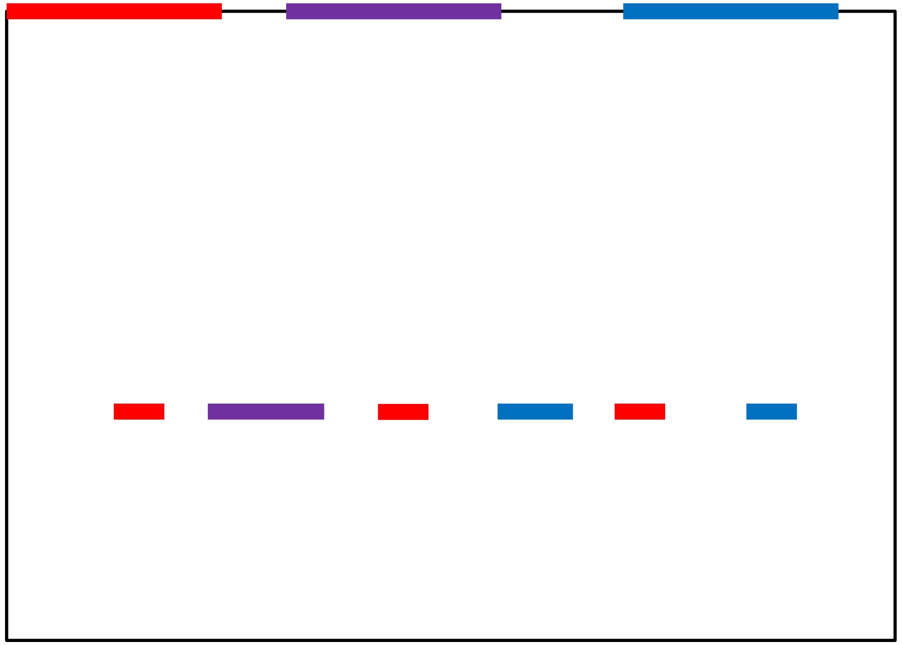


Recursive Partitioning

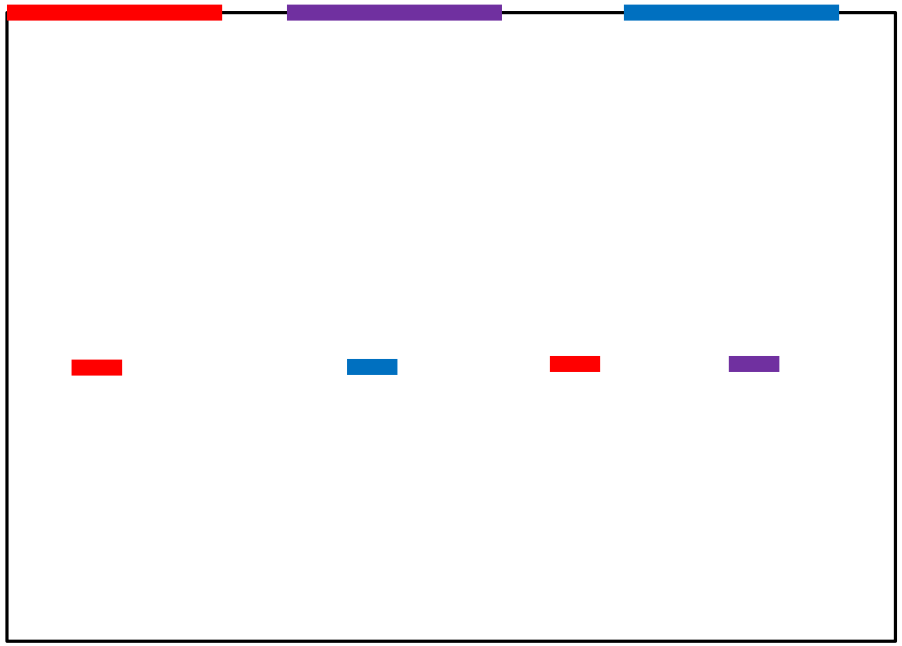
Theorem: Recursive Partitioning Property holds for x demand pairs $\Rightarrow$ can route all x demand pairs

Routing in two parts

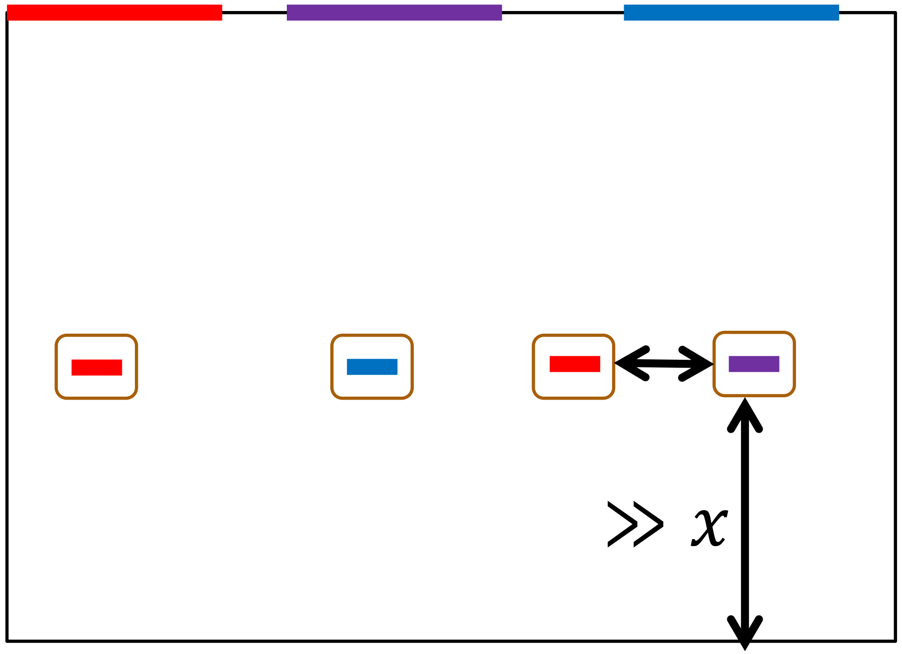
- "Global Routing"
- "Local Routing"



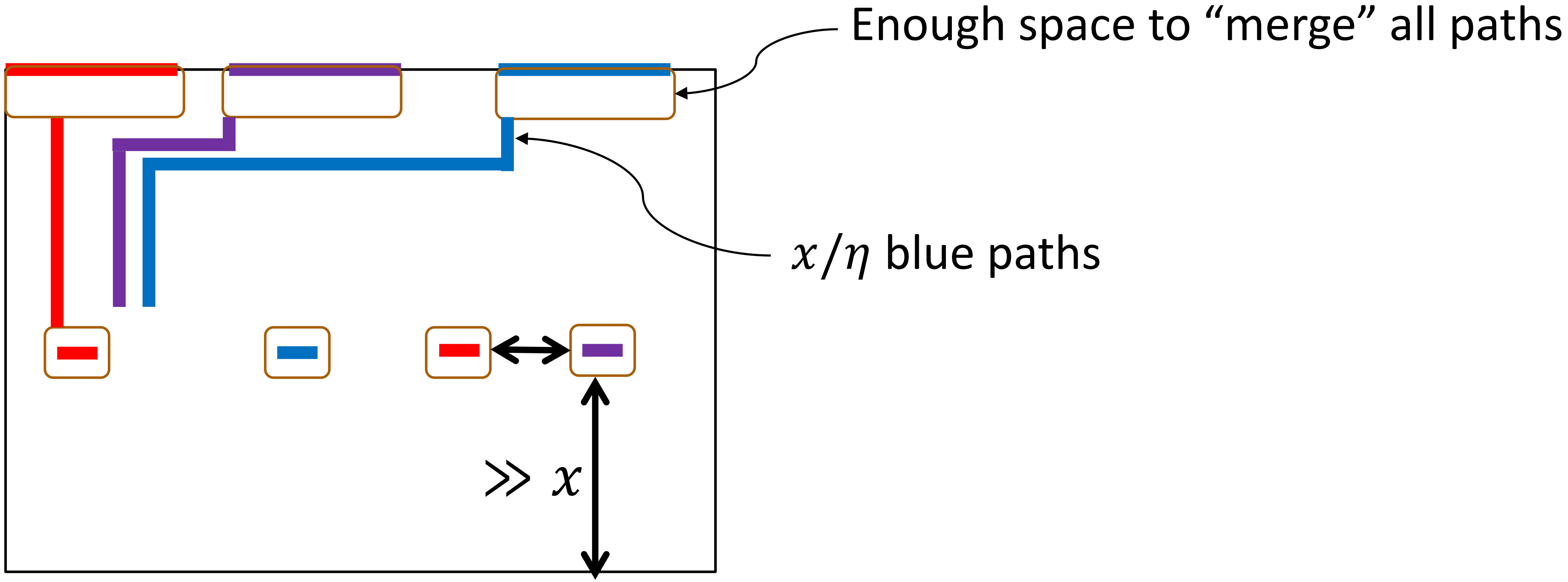
Part 1 | Global Routing



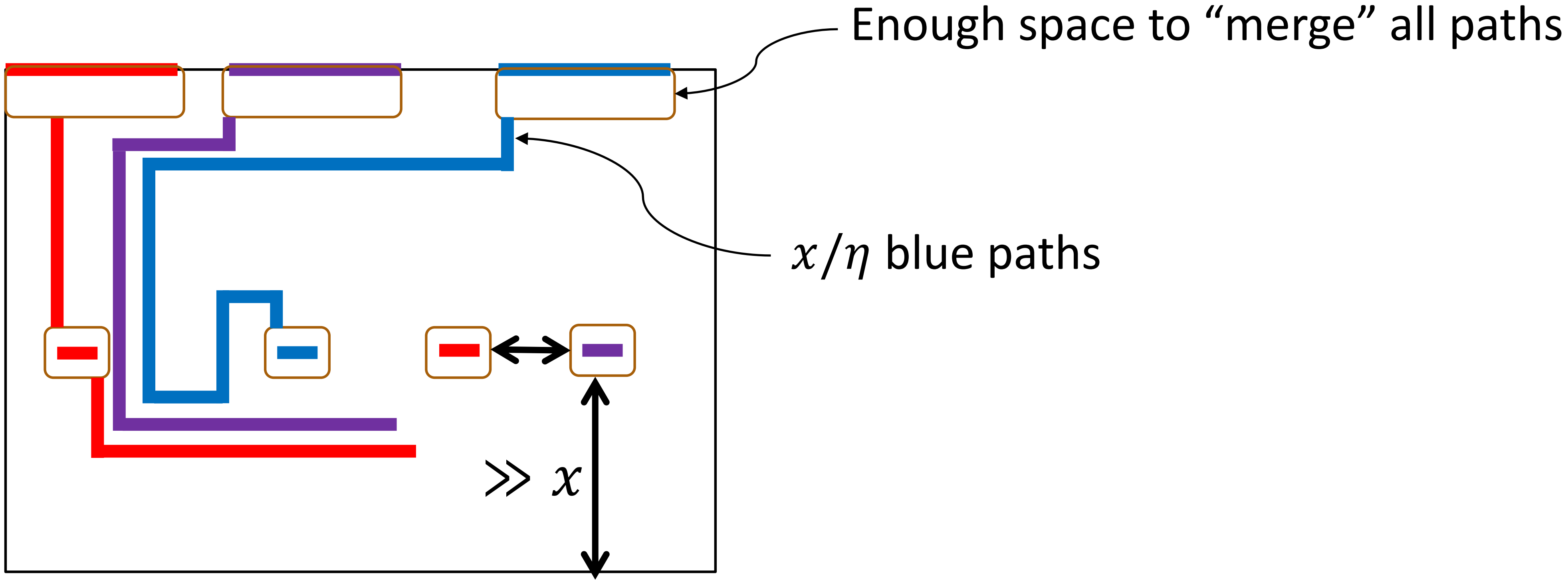
Part 1 | Global Routing



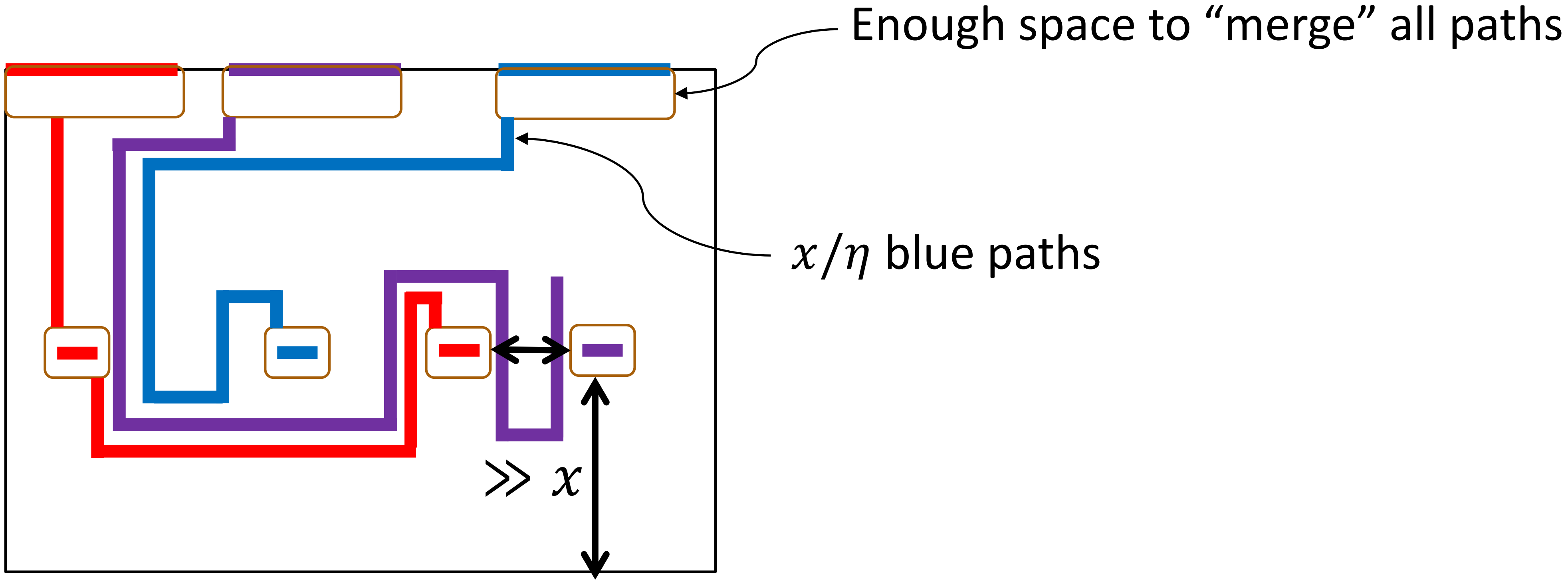
Part 1 | Global Routing



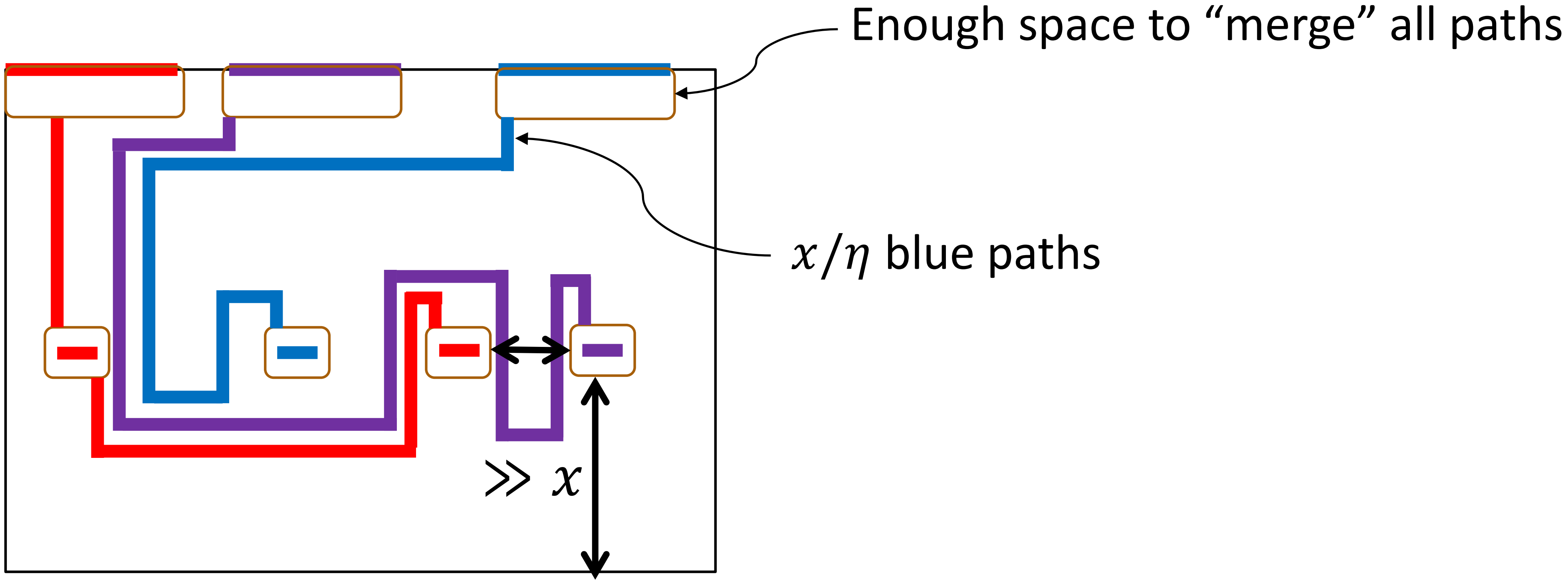
Part 1 | Global Routing



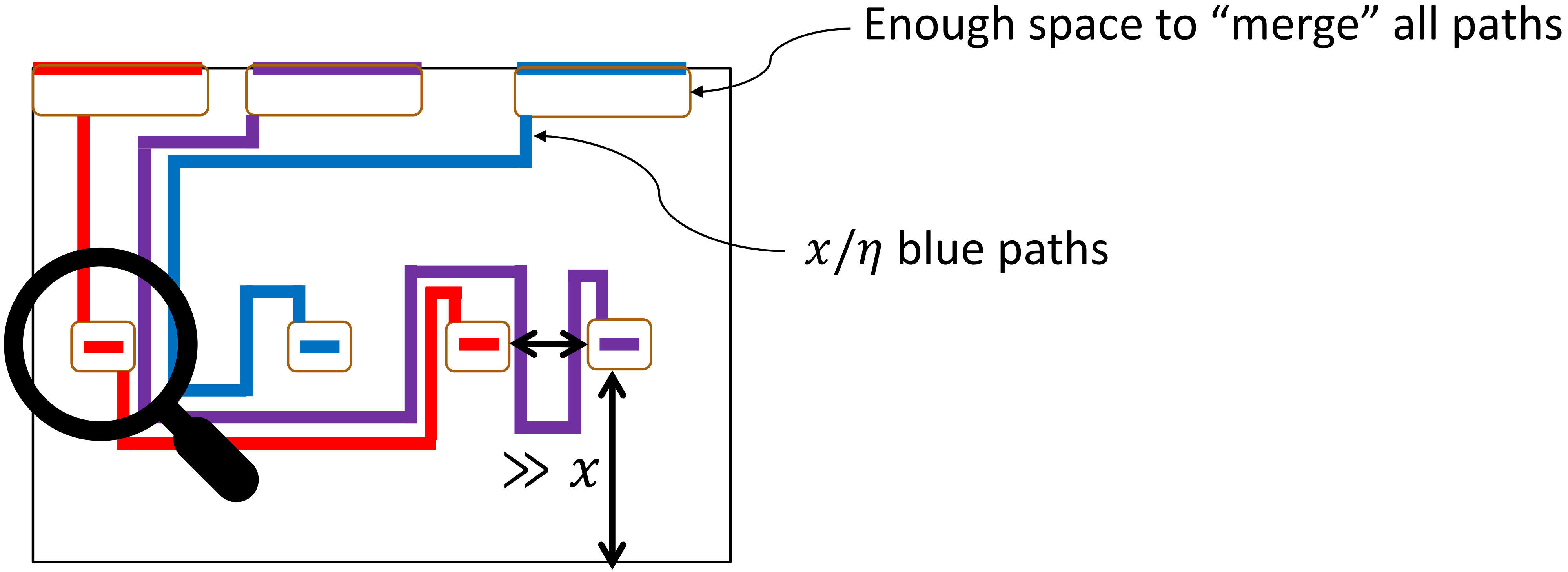
Part 1 | Global Routing



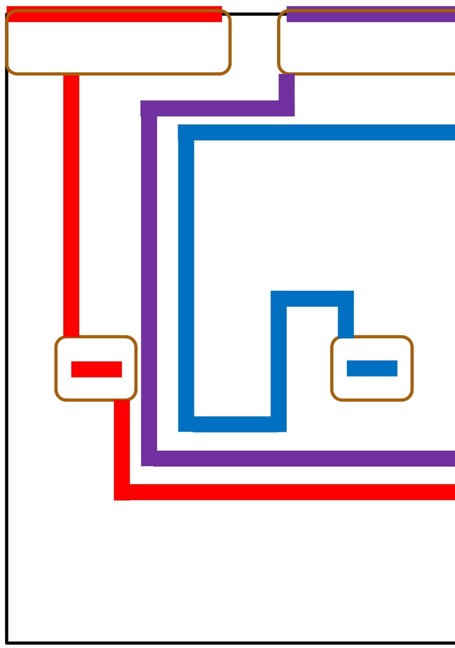
Part 1 | Global Routing



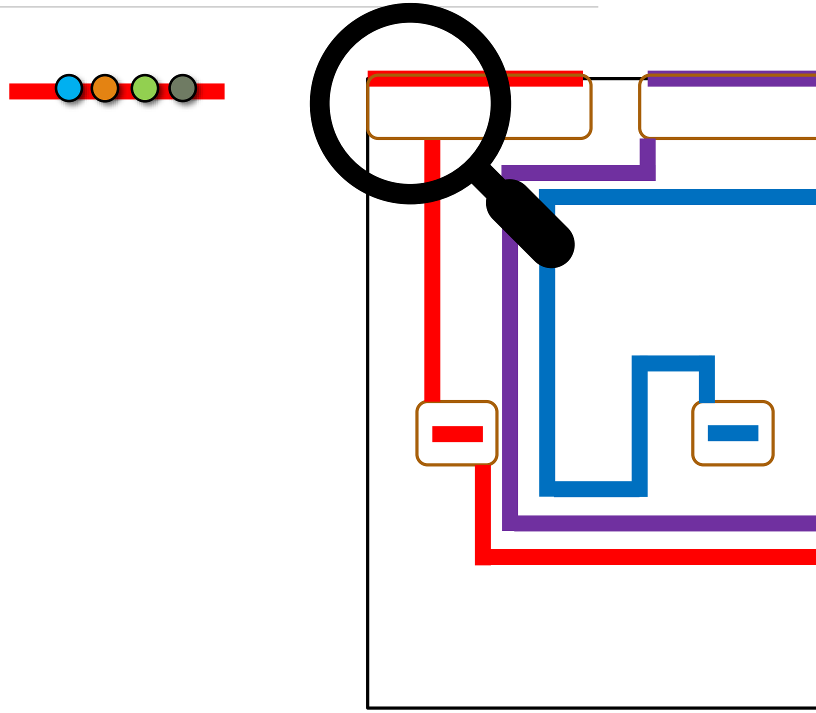
Part 1 | Global Routing



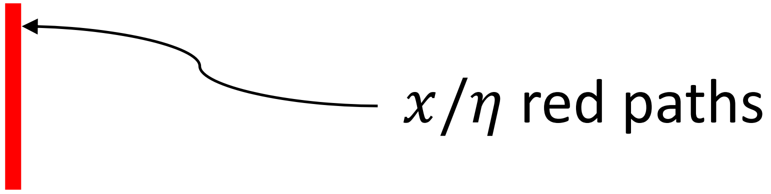
Part 2 | Local Routing

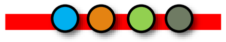


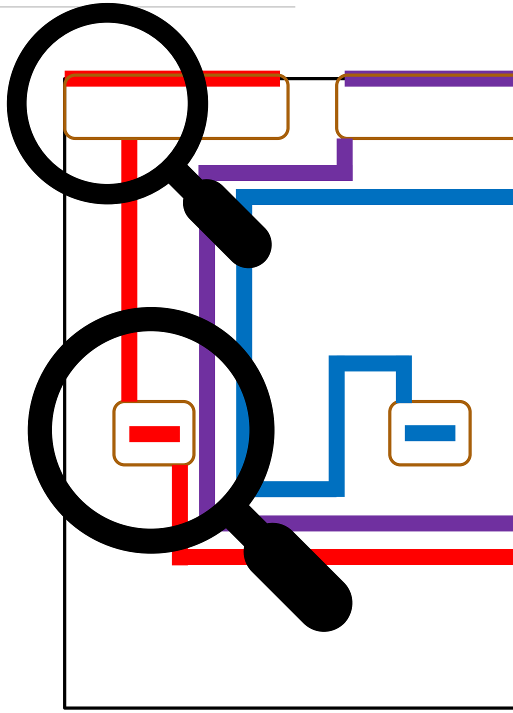
Part 2 | Local Routing



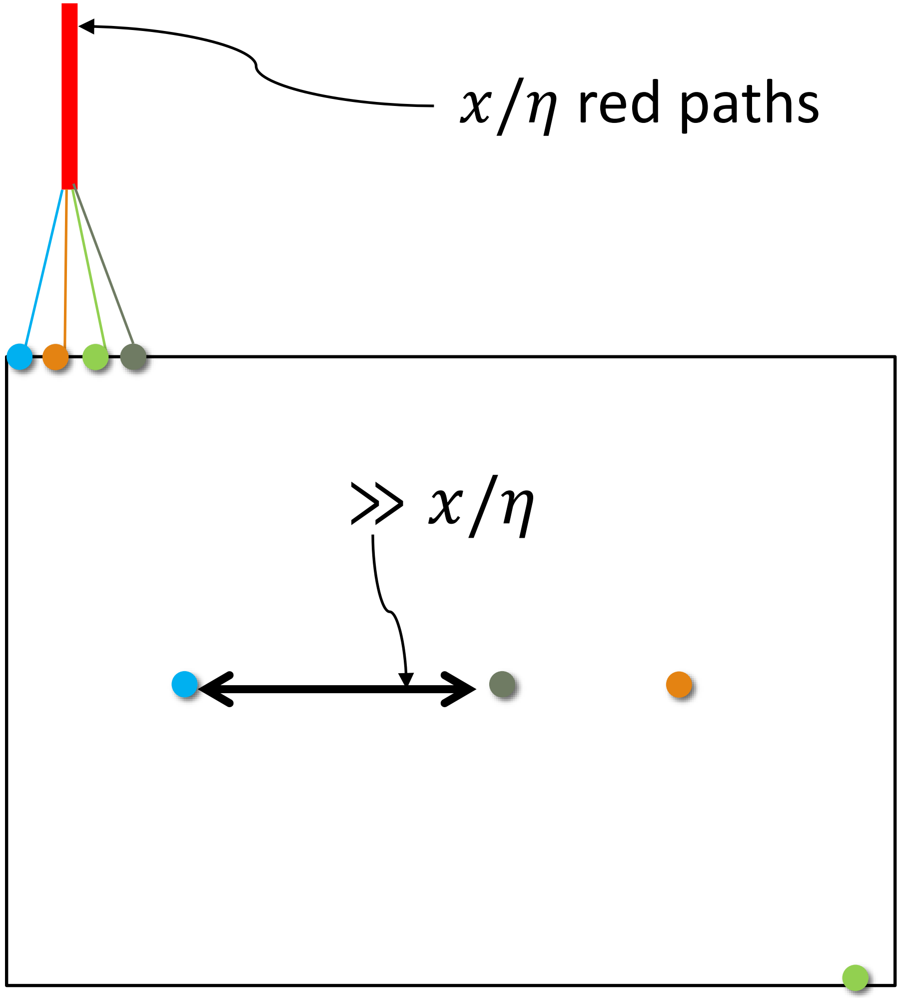
Part 2 | Local Routing

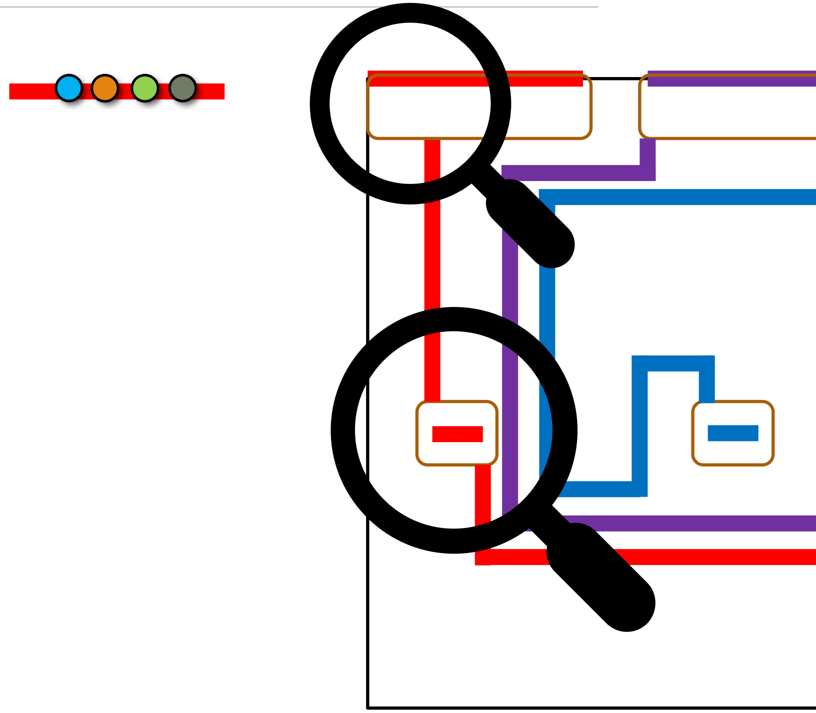




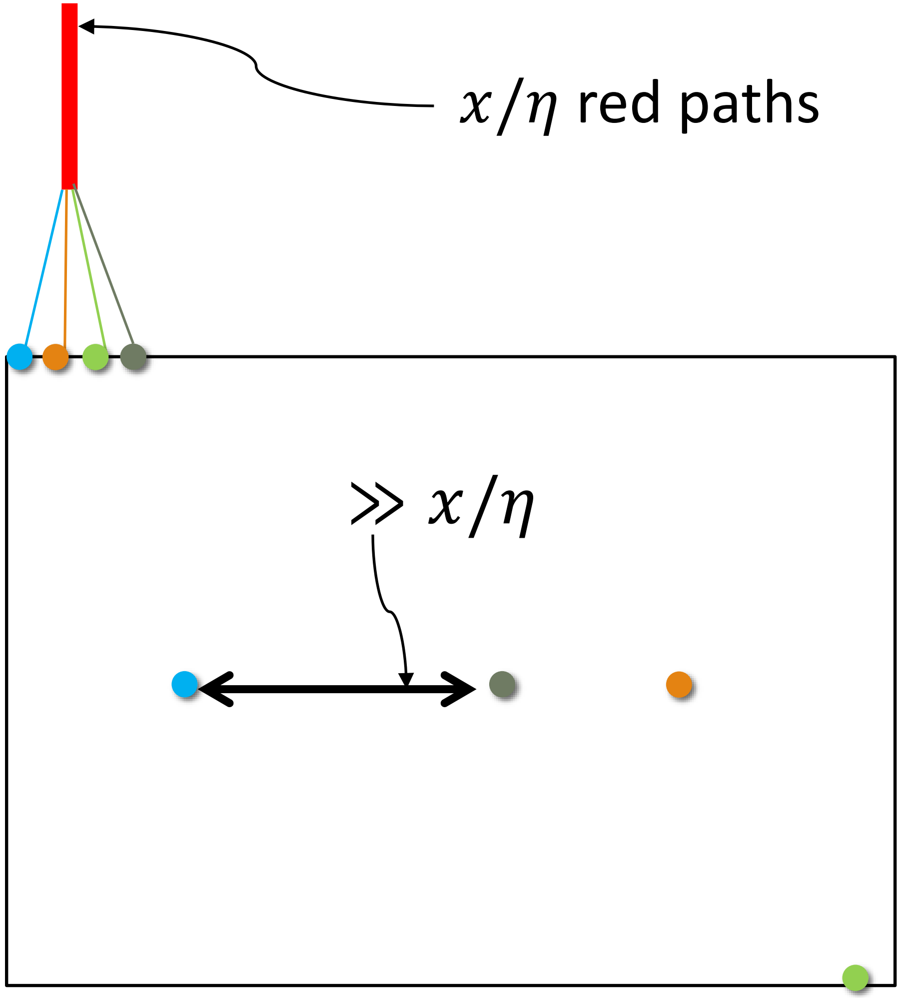


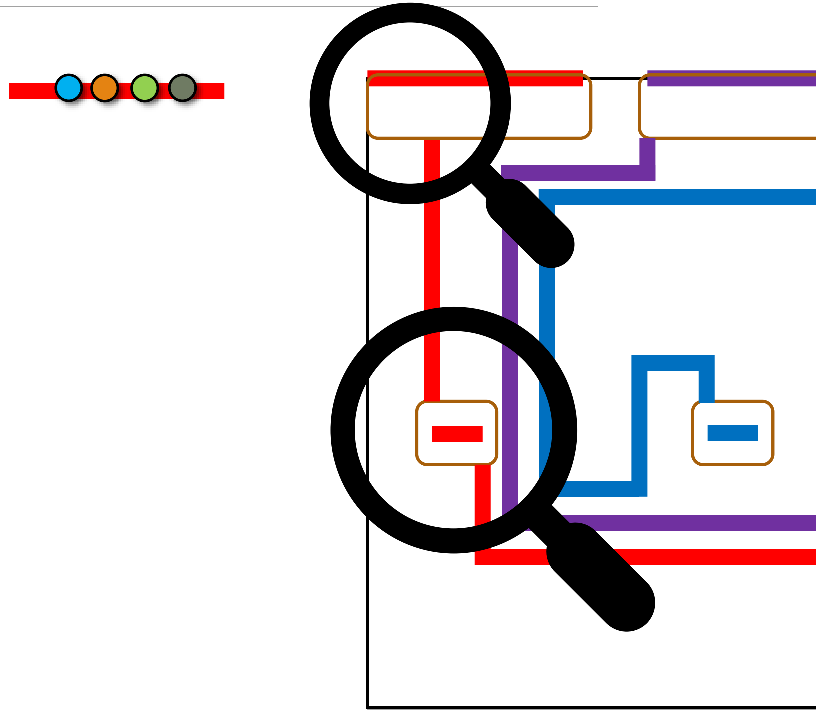
Part 2 | Local Routing





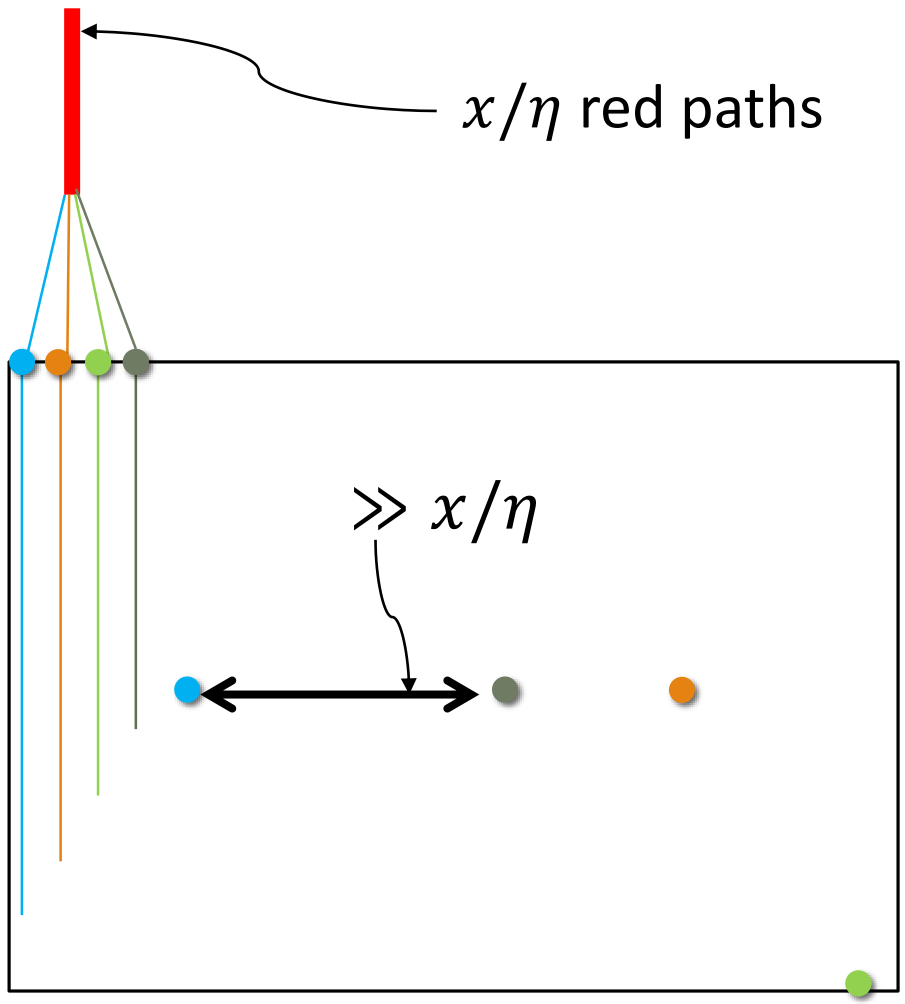
Part 2 | Local Routing

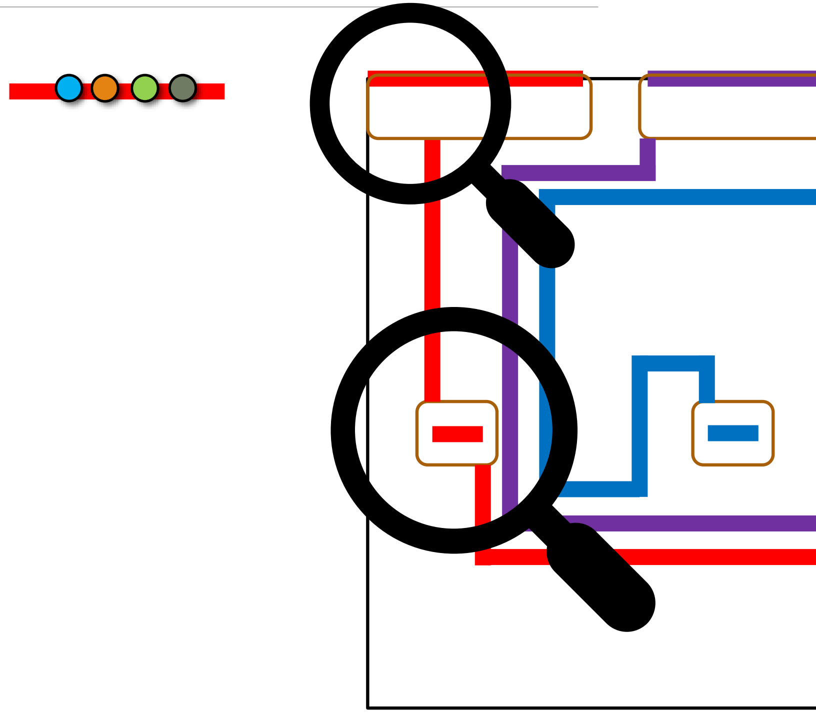




Snake-like routing
that we saw earlier!

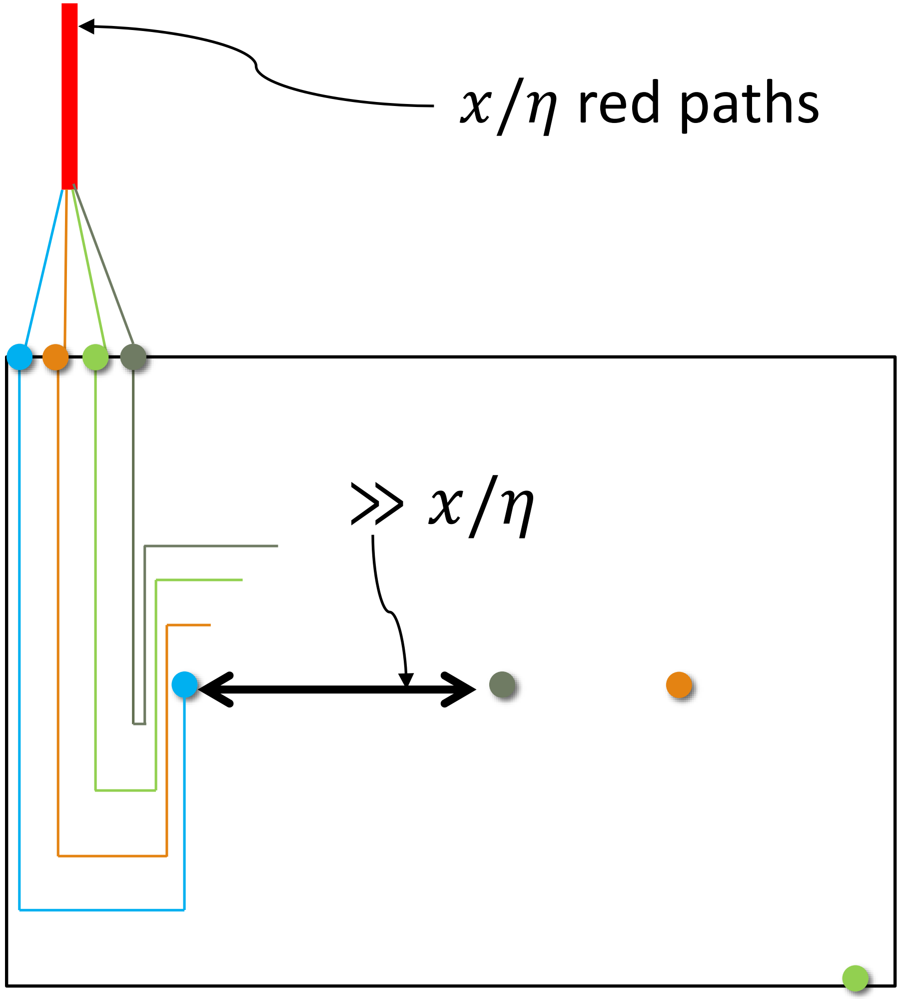
Part 2 | Local Routing

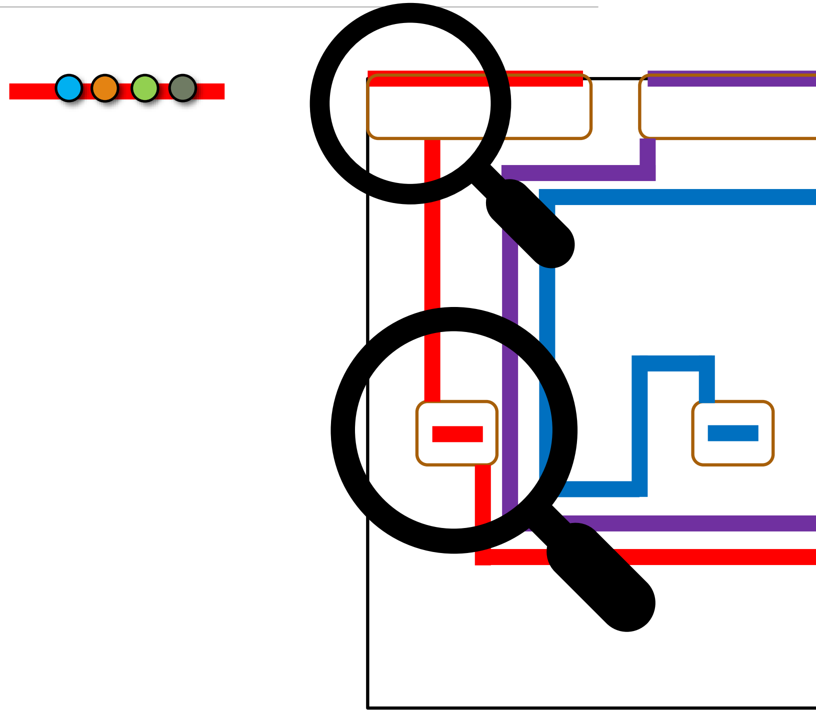




Snake-like routing
that we saw earlier!

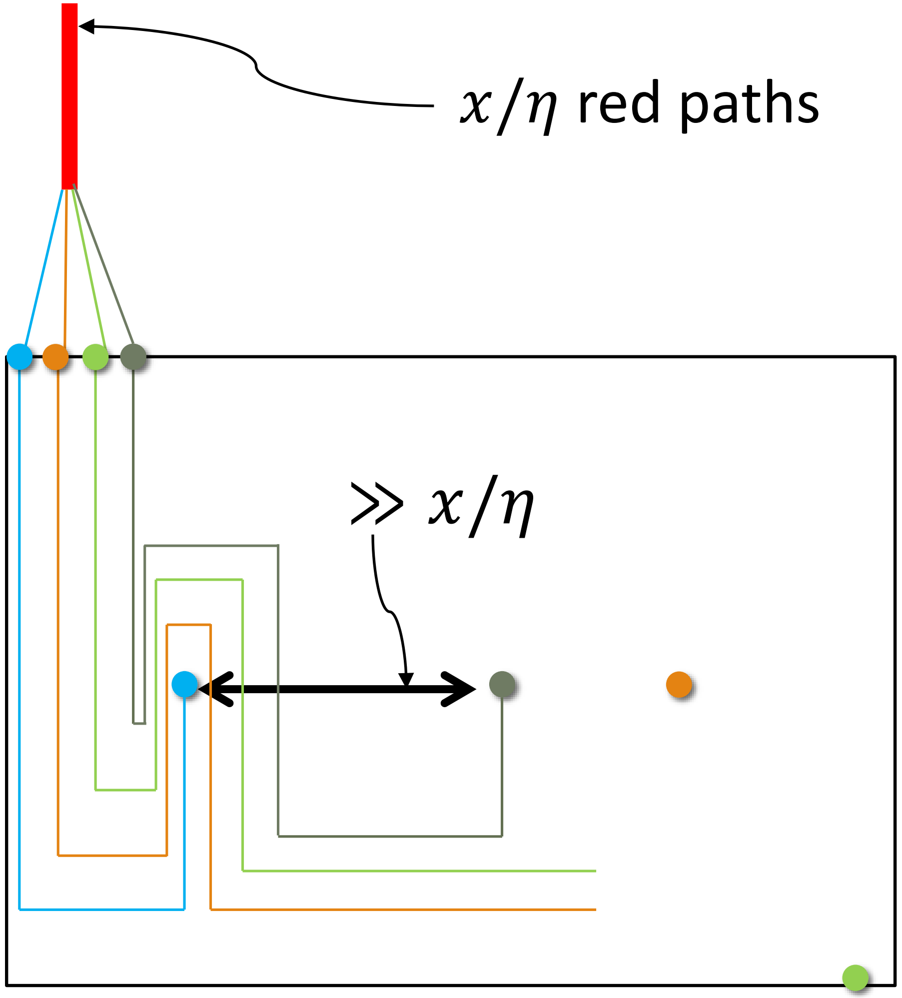
Part 2 | Local Routing

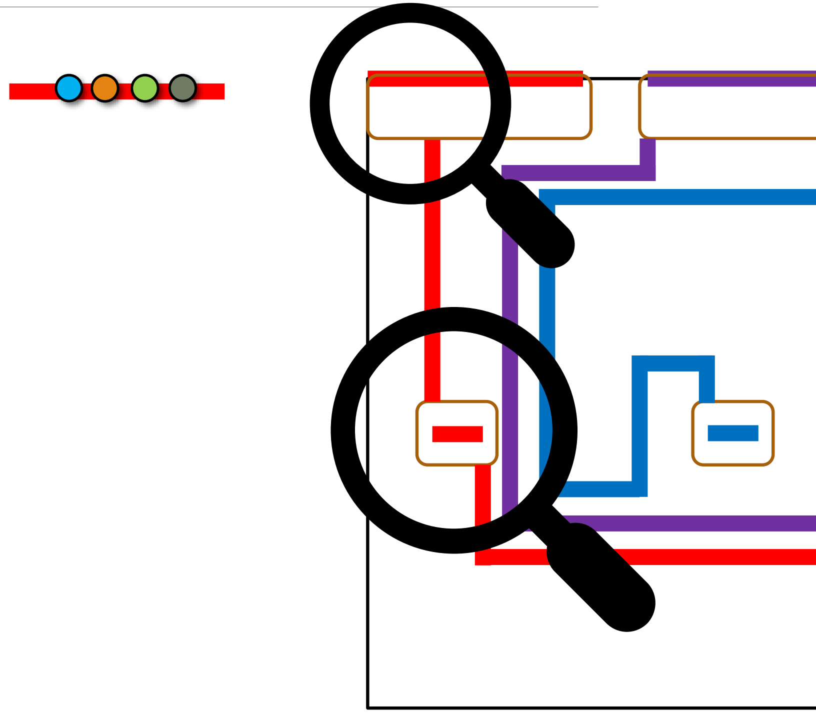




Snake-like routing
that we saw earlier!

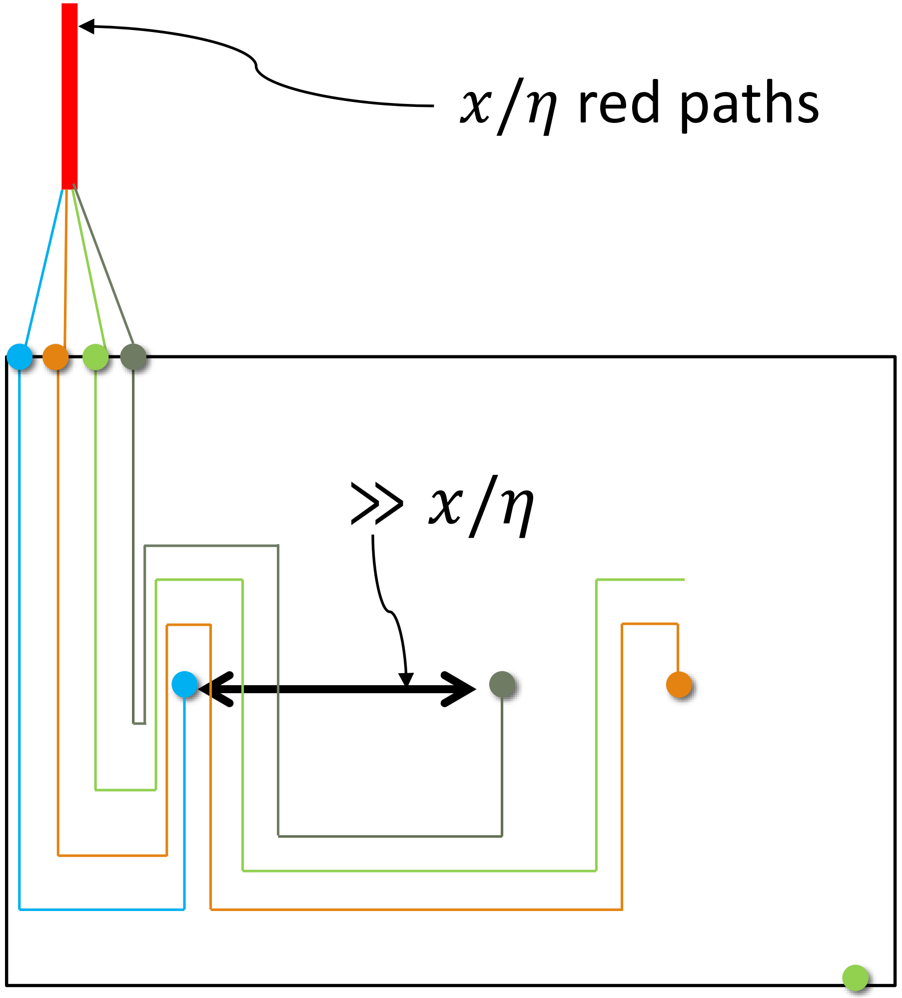
Part 2 | Local Routing

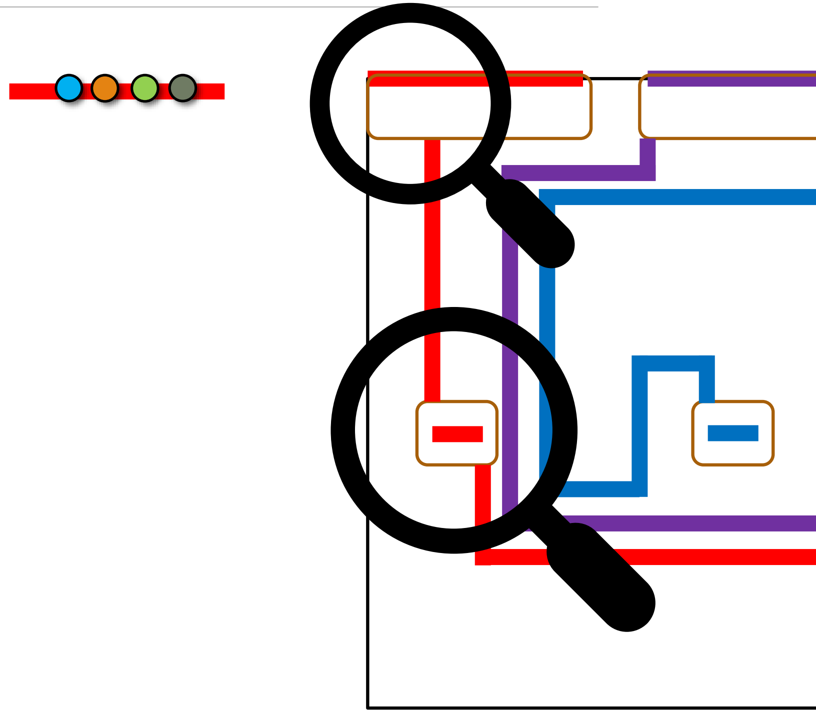




Snake-like routing
that we saw earlier!

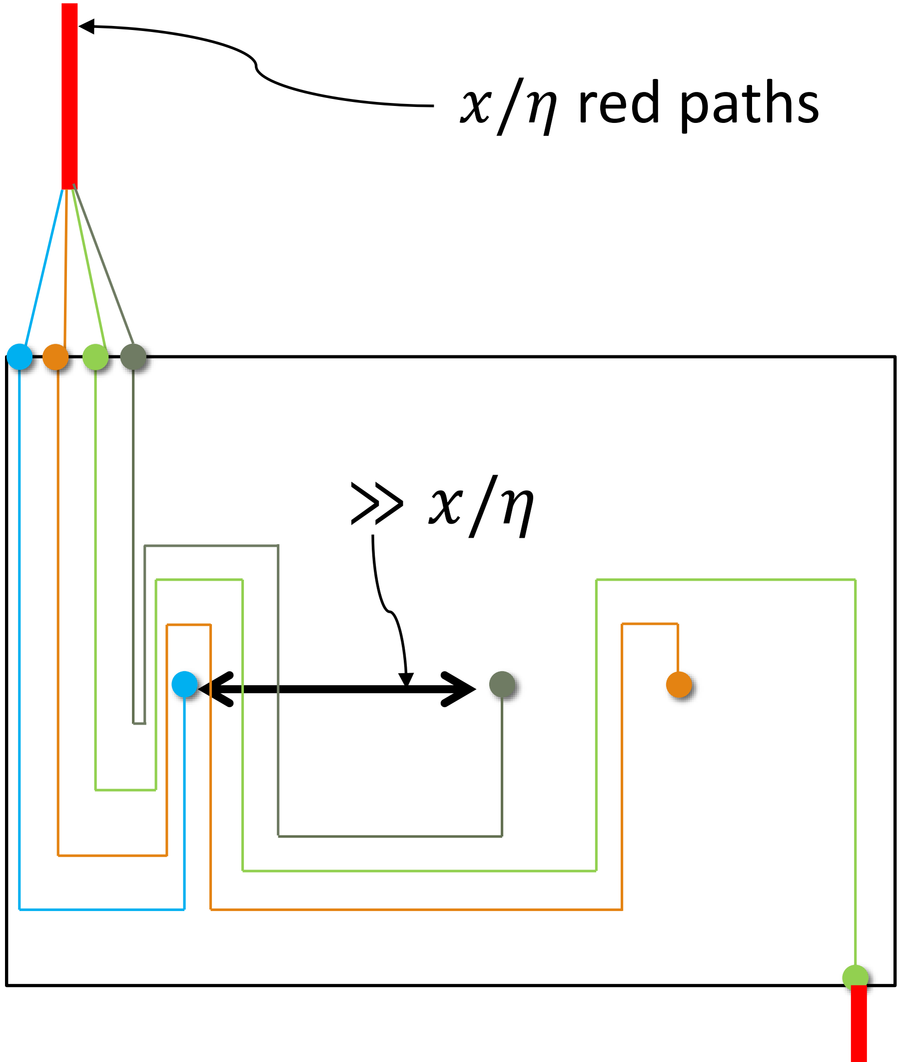
Part 2 | Local Routing

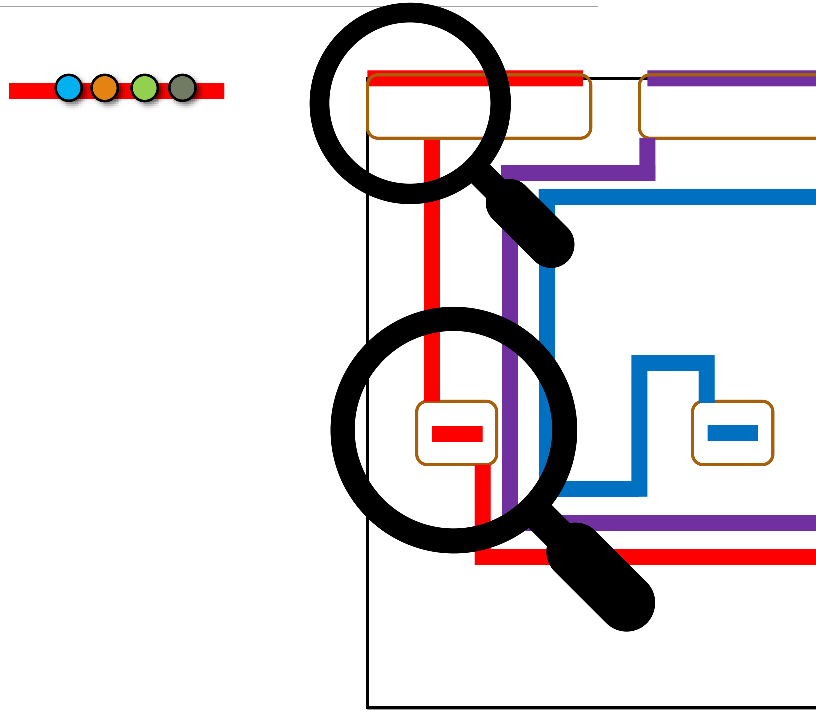




Snake-like routing
that we saw earlier!

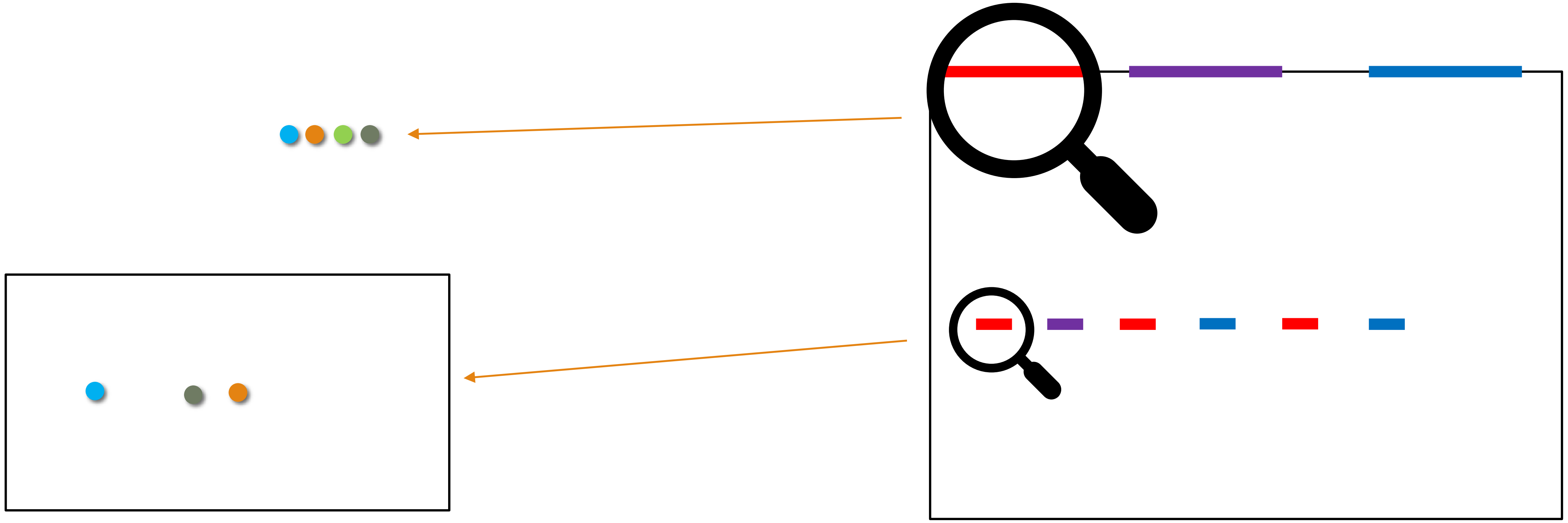
Part 2 | Local Routing





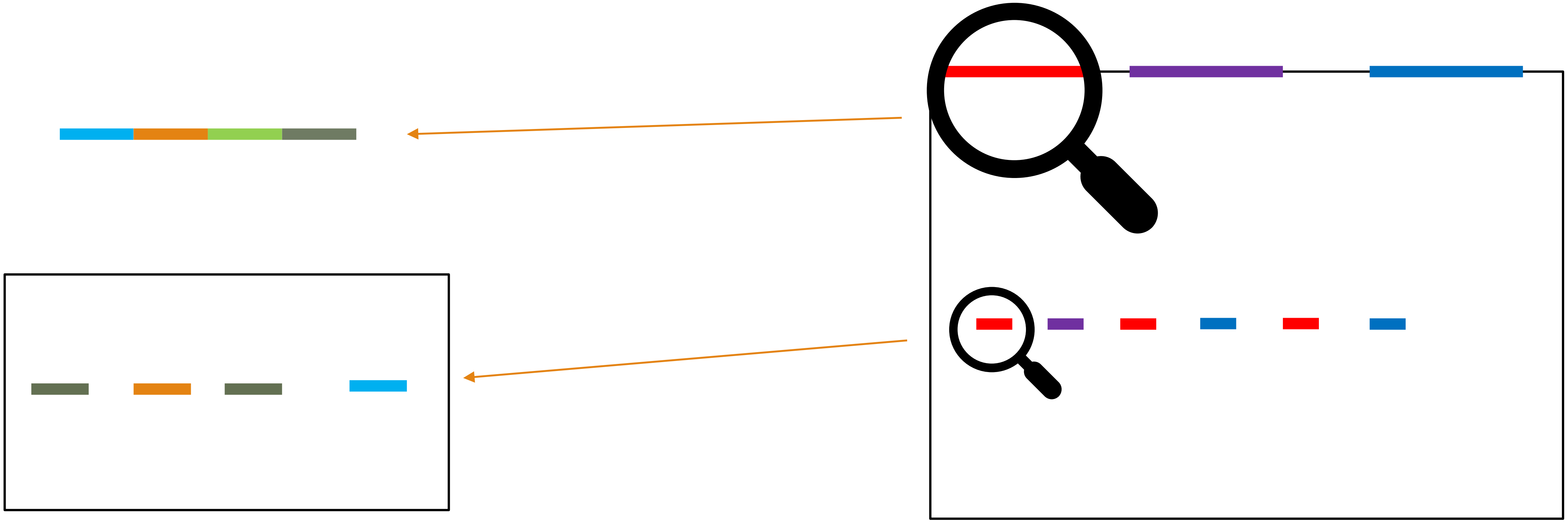
Snake-like routing
that we saw earlier!

Recursive Partitioning



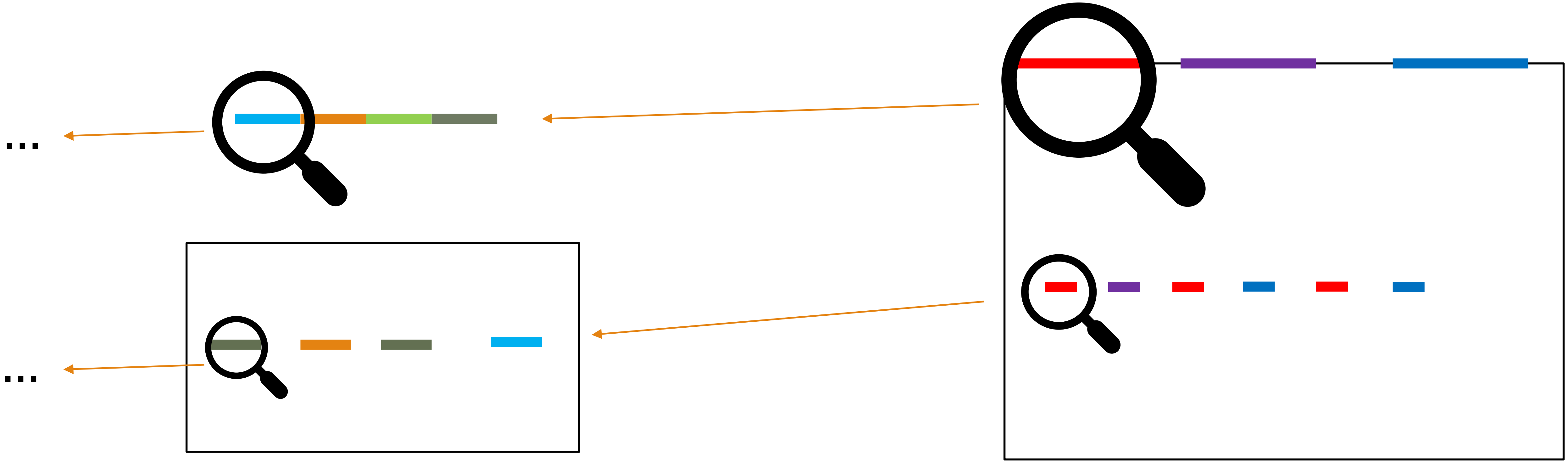
Recursive Partitioning

To optimize our approximation ratio, we set $\eta = 2^{\sqrt{\log n}}$ and extend this approach to $\sqrt{\log n}$ levels



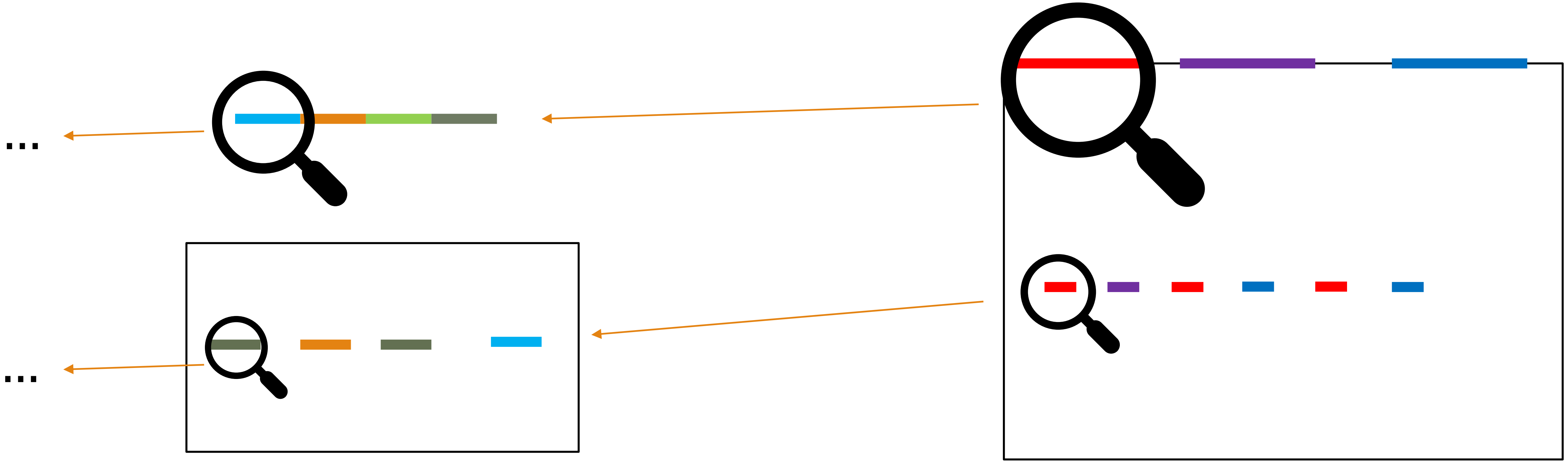
Recursive Partitioning

To optimize our approximation ratio, we set $\eta = 2^{\sqrt{\log n}}$ and extend this approach to $\sqrt{\log n}$ levels



Recursive Partitioning

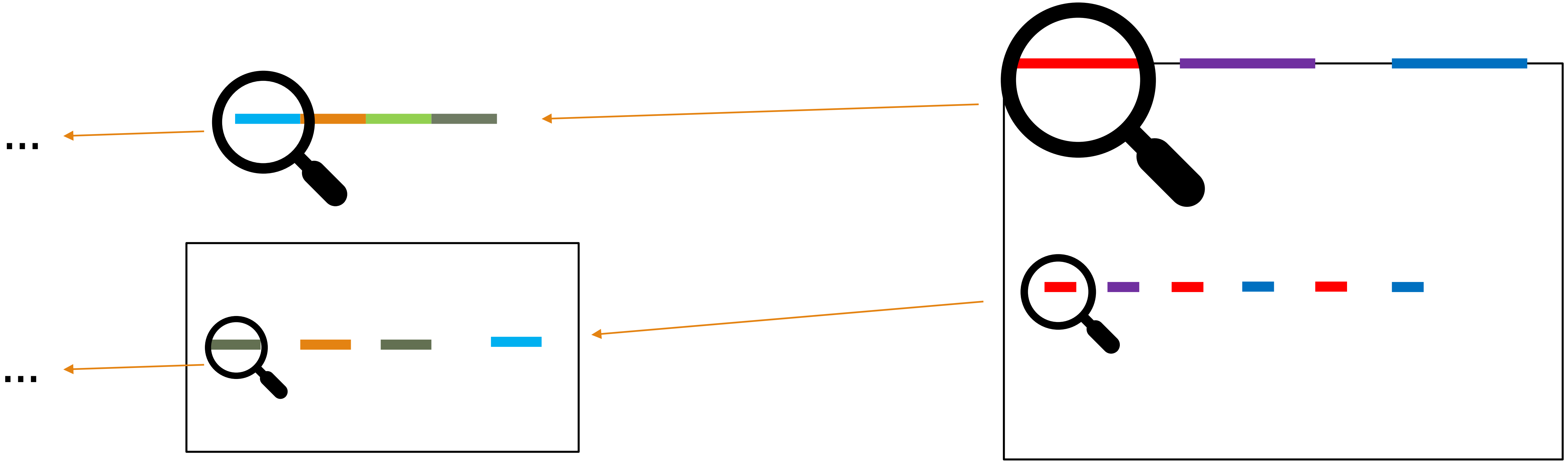
Recursive Partitioning = “*hierarchical set of intervals*” and “*hierarchical assignment*” of colors



Recursive Partitioning

Recursive Partitioning = “*hierarchical set of intervals*” and “*hierarchical assignment*” of colors

Nested intervals on source-row and nested intervals on destination-row

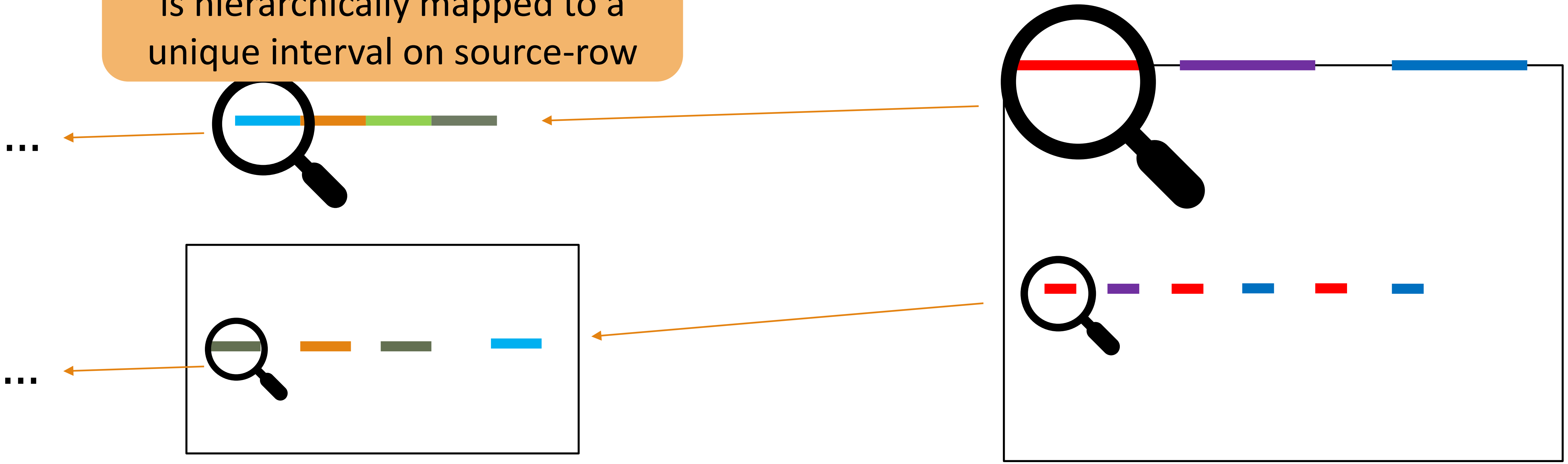


Recursive Partitioning

Recursive Partitioning = “*hierarchical set of intervals*” and “*hierarchical assignment*” of colors

Each interval on destination-row is hierarchically mapped to a unique interval on source-row

Nested intervals on source-row and nested intervals on destination-row



Recursive Partitioning | Why?

Theorem: $OPT_{RP} \geq OPT/2^{\tilde{O}(\sqrt{\log n})}$

Largest subset of demand pairs with Recursive Partitioning Property

Value of the optimum NDP-Grid solution

Recursive Partitioning | How?

Theorem: Can efficiently find a set of $OPT_{RP}/2^{O(\sqrt{\log n})}$ demand pairs with Recursive Partitioning Property

Recursive Partitioning | How?

Theorem: Can efficiently find a set of $OPT_{RP}/2^{O\left(\sqrt{\log n}\right)}$ demand pairs with Recursive Partitioning Property

Idea:

- Find a small collection of candidate hierarchical sets of intervals such that one of them has this property
- Solve for each candidate separately
- Return the best solution

Recursive Partitioning | How?

Theorem: Can efficiently find a set of $OPT_{RP}/2^{O\left(\sqrt{\log n}\right)}$ demand pairs with Recursive Partitioning Property

Idea:

- Find a small collection of candidate hierarchical sets of intervals such that one of them has this property
- Solve for each candidate separately
- Return the best solution

Write LP and perform randomized rounding level by level

Final Algorithm

- Find a set of OPT_{RP} / $2^{O\left(\sqrt{\log n}\right)}$ demand pairs with Recursive Partitioning Property
- Route all of them!

Final Algorithm

- Find a set of $OPT_{RP}/\ 2^{O\left(\sqrt{\log n}\right)}$ demand pairs with Recursive Partitioning Property
- Route all of them!

Recall: $OPT_{RP} \geq OPT/2^{\tilde{O}\left(\sqrt{\log n}\right)}$

We route $OPT/2^{\tilde{O}\left(\sqrt{\log n}\right)}$ demand pairs

Conclusion

- $2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP-Grids if sources appear on the boundary

Conclusion

- $2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP-Grids if sources appear on the boundary
- $\delta \cdot 2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP- Grids if sources are at distance $\leq \delta$ from the boundary

Conclusion

- $2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP-Grids if sources appear on the boundary
- $\delta \cdot 2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP- Grids if sources are at distance $\leq \delta$ from the boundary
- Only APX-hardness is known for NDP-Grid with sources on the boundary

  Better hardness results for this case?
- Congestion minimization?
  - Route everything, but minimize *load*
  - $O(\log n / \log\log n)$-approx
  - $\Omega(\log\log n)$-hardness

Conclusion

- $2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP-Grids if sources appear on the boundary
- $\delta \cdot 2^{\tilde{O}(\sqrt{\log n})}$-approximation algorithm for NDP- Grids if sources are at distance $\leq \delta$ from the boundary
- Only APX-hardness is known for NDP-Grid with sources on the boundary

  Better hardness results for this case?
- Congestion minimization?
  - Route everything, but minimize *load*
  - $O(\log n / \log \log n)$-approx
  - $\Omega(\log \log n)$-hardness

Thank You!